CS162
Operating Systems and Systems Programming
Lecture 27

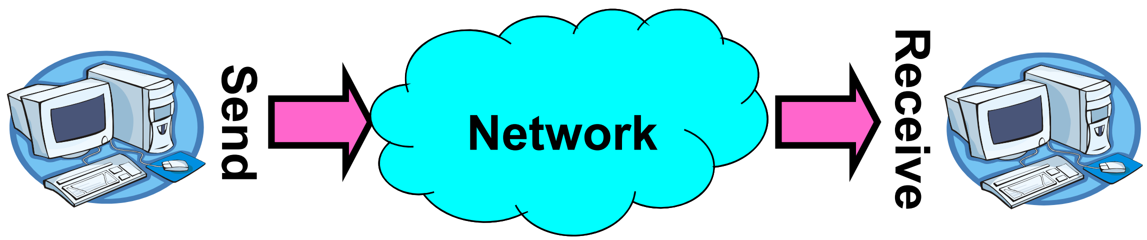
Distributed File Systems
Quantum Computing

May 2nd, 2023

Prof. John Kubiatowicz
http://cs162.eecs.Berkeley.edu

Recall: Distributed Applications Build With Messages

- How do you actually program a distributed application?
  - Need to synchronize multiple threads, running on different machines
    - » No shared memory, so cannot use test&set



  - One Abstraction: send/receive messages
    - » Already atomic: no receiver gets portion of a message and two receivers cannot get same message
- Interface:
  - Mailbox (mbox): temporary holding area for messages
    - » Includes both destination location and queue
  - Send(message,mbox)
    - » Send message to remote mailbox identified by mbox
  - Receive(buffer,mbox)
    - » Wait until mbox has message, copy into buffer, and return
    - » If threads sleeping on this mbox, wake up one of them

Recall: Endianness

- For a byte-address machine, which end of a machine-recognized object (e.g., int) does its byte-address refer to?
- Big Endian: address is the most-significant bits
- Little Endian: address is the least-significant bits

Processor	Endianness
Motorola 68000	Big Endian
PowerPC (PPC)	Big Endian
Sun Sparc	Big Endian
IBM S/390	Big Endian
Intel x86 (32 bit)	Little Endian
Intel x86_64 (64 bit)	Little Endian
Dec VAX	Little Endian
Alpha	Bi (Big/Little) Endian
ARM	Bi (Big/Little) Endian
IA-64 (64 bit)	Bi (Big/Little) Endian
MIPS	Bi (Big/Little) Endian

```
int main(int argc, char *argv[])
{
  int val = 0x12345678;
  int i;
  printf("val = %x\n", val);
  for (i = 0; i < sizeof(val); i++) {
    printf("val[%d] = %x\n", i, ((uint8_t *) &val)[i]);
  }
}
```

```
(base) CullerMac19:code09 culler$ ./endian
val = 12345678
val[0] = 78
val[1] = 56
val[2] = 34
val[3] = 12
```

Dealing with Endianness between Hosts

- Decide on an “on-wire” endianness
- Convert from native endianness to “on-wire” endianness before sending out data (**serialization/marshalling**)
  - `uint32_t htonl(uint32_t)` and `uint16_t htons(uint16_t)` convert from native endianness to network endianness (big endian)
- Convert from “on-wire” endianness to native endianness when receiving data (**deserialization/unmarshalling**)
  - `uint32_t ntohl(uint32_t)` and `uint16_t ntohs(uint16_t)` convert from network endianness to native endianness (big endian)
- What “endianness” is the network?
  - Big Endian
  - Network macros (`htonl()`, `htons()`, `ntohl()`, and `ntohs()`) convert for you without you needing to know one way or another.

What About Richer Objects?

```
typedef struct word_count
{
  char *word;
  int count;
  struct word_count *next;
}
word_count_t;
```

- Consider `word_count_t` of Homework 0 and 1 …
- Each element contains:
  – An `int`
  – A *pointer* to a string (of some length)
  – A *pointer* to the next element
- `fprintf_words` writes these as sequence of lines (character strings with `\n`) to a file
- What if you wanted to write the whole list as a binary object (and read it back as one)?
  – How do you represent the string?
  – Does it make any sense to write the pointer?
- Marshalling involves (depending on system)
  – Converting values to a canonical form, serializing objects, copying arguments passed by reference, etc.
  – Also called “serialization”
- Unmarshaling involves
  – Reconstructing the original object from its marshalled form at destination
  – Also called “deserialization”

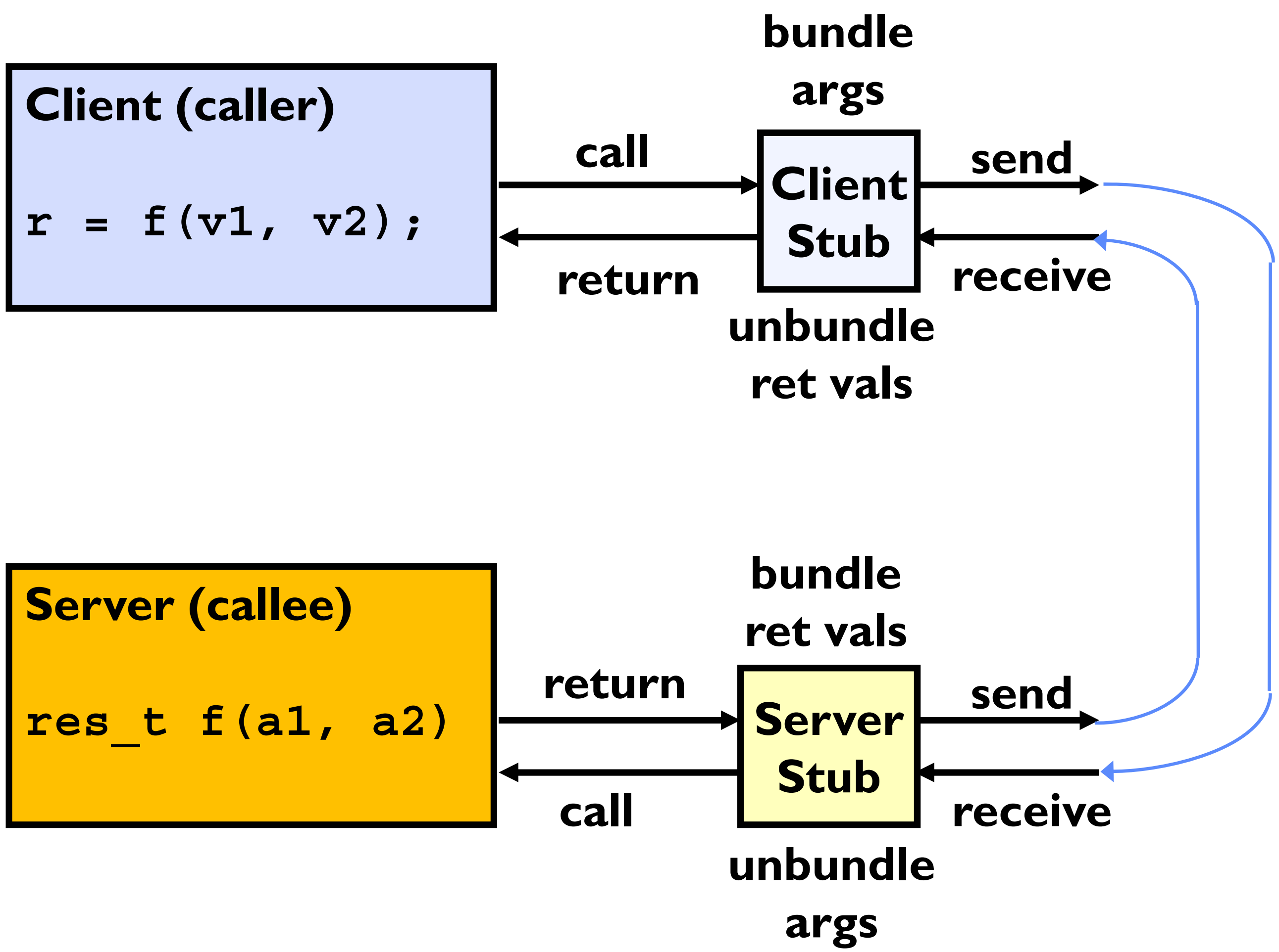
Data Serialization Formats (MANY!)

Name	Creator-maintainer	Based on	Standardized?	Specification	Binary?	Human-readable?	Supports references?[e]	Schema-IDL?	Standard APIs	Supports Zero-copy operations
Apache Avro	Apache Software Foundation	N/A	No	Apache Avro™ 1.8.1 Specification	Yes	No	N/A	Yes (built-in)	N/A	N/A
Apache Parquet	Apache Software Foundation	N/A	No	Apache Parquet[1]	Yes	No	No	N/A	Java, Python	No
ASN.1	ISO, IEC, ITU-T	N/A	Yes	ISO/IEC 8824; X.680 series of ITU-T Recommendations	Yes (BER, DER, PER, OER, or custom via ECN)	Yes (XER, JER, GSER, or custom via ECN)	Partial[f]	Yes (built-in)	N/A	Yes (OER)
Bencode	Bram Cohen (creator) BitTorrent, Inc. (maintainer)	N/A	*De facto* standard via BitTorrent Enhancement Proposal (BEP)	Part of BitTorrent protocol specification	Partially (numbers and delimiters are ASCII)	No	No	No	No	N/A
Binn	Bernardo Ramos	N/A	No	Binn Specification	Yes	No	No	No	No	Yes
BSON	MongoDB	JSON	No	BSON Specification	Yes	No	No	No	No	N/A
CBOR	Carsten Bormann, P. Hoffman	JSON (loosely)	Yes	RFC 7049	Yes	No	Yes through tagging	Yes (CDDL)	No	Yes
Comma-separated values (CSV)	RFC author: Yakov Shafranovich	N/A	Partial (myriad informal variants used)	RFC 4180 (among others)	No	Yes	No	No	No	No
Common Data Representation (CDR)	Object Management Group	N/A	Yes	General Inter-ORB Protocol	Yes	No	Yes	Yes	ADA, C, C++, Java, Cobol, Lisp, Python, Ruby, Smalltalk	N/A
D-Bus Message Protocol	freedesktop.org	N/A	Yes	D-Bus Specification	Yes	No	No	Partial (Signature strings)	Yes (see D-Bus)	N/A
Efficient XML Interchange (EXI)	W3C	XML, Efficient XML	Yes	Efficient XML Interchange (EXI) Format 1.0	Yes	Yes (XML)	Yes (XPointer, XPath)	Yes (XML Schema)	Yes (DOM, SAX, StAX, XQuery, XPath)	N/A
FlatBuffers	Google	N/A	No	flatbuffers github page Specification	Yes	Yes (Apache Arrow)	Partial (internal to the buffer)	Yes [2]	C++, Java, C#, Go, Python, Rust, JavaScript, PHP, C, Dart, Lua, TypeScript	Yes
Fast Infoset	ISO, IEC, ITU-T	XML	Yes	ITU-T X.891 and ISO/IEC 24824-1:2007	Yes	No	Yes (XPointer, XPath)	Yes (XML schema)	Yes (DOM, SAX, XQuery, XPath)	N/A
FHIR	Health_Level_7	REST basics	Yes	Fast Healthcare Interoperability Resources	Yes	Yes	Yes	Yes	Hapi for FHIR[1] JSON, XML, Turtle	No
Ion	Amazon	JSON	No	The Amazon Ion Specification	Yes	Yes	No	No	No	N/A
Java serialization	Oracle Corporation	N/A	Yes	Java Object Serialization	Yes	No	Yes	No	Yes	N/A
JSON	Douglas Crockford	JavaScript syntax	Yes	STD 90/RFC 8259 (ancillary: RFC 6901, RFC 6902), ECMA-404, ISO/IEC 21778:2017	No, but see BSON, Smile, UBJSON	Yes	Yes (JSON Pointer (RFC 6901); alternately: JSONPath, JPath, JSPON, json:select()), JSON-LD	Partial (JSON Schema Proposal, ASN.1 with JER, Kwalify, Rx, Itemscript Schema), JSON-LD	Partial (Clarinet, JSONQuery, JSONPath), JSON-LD	No
MessagePack	Sadayuki Furuhashi	JSON (loosely)	No	MessagePack format specification	Yes	No	No	No	No	Yes
Netstrings	Dan Bernstein	N/A	No	netstrings.txt	Yes	Yes	No	No	No	Yes
OGDL	Rolf Veen	?	No	Specification	Yes (Binary Specification)	Yes	Yes (Path Specification)	Yes (Schema WD)		N/A
OPC-UA Binary	OPC Foundation	N/A	No	opcfoundation.org	Yes	No	Yes	No	No	N/A
OpenDDL	Eric Lengyel	C, PHP	No	OpenDDL.org	No	Yes	Yes	No	Yes (OpenDDL Library)	N/A
Pickle (Python)	Guido van Rossum	Python	*De facto* standard via Python Enhancement Proposals (PEPs)	[3] PEP 3154 -- Pickle protocol version 4	Yes	No	No	No	Yes ([4])	No
Property list	NeXT (creator) Apple (maintainer)	?	Partial	Public DTD for XML format	Yes[a]	Yes[b]	No	?	Cocoa, CoreFoundation, OpenStep, GnuStep	No
Protocol Buffers (protobuf)	Google	N/A	No	Developer Guide: Encoding	Yes	Partial[d]	No	Yes (built-in)	C++, C#, Java, Python, Javascript, Go	No
	John McCarthy					Yes				

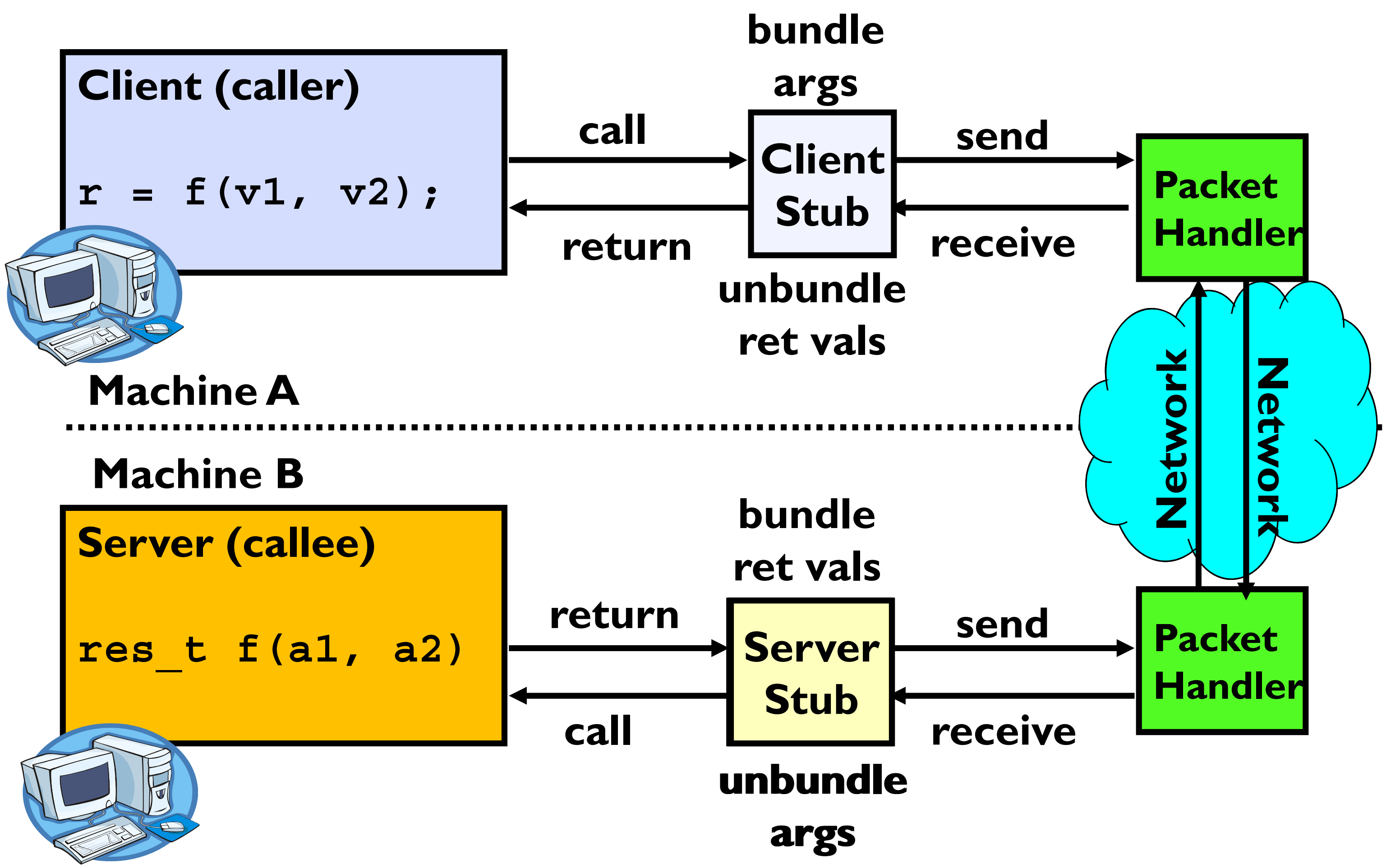
Remote Procedure Call (RPC)

- Raw messaging is a bit too low-level for programming
  - Must wrap up information into message at source
  - Must decide what to do with message at destination
  - May need to sit and wait for multiple messages to arrive
  - And must deal with machine representation by hand
- Another option: Remote Procedure Call (RPC)
  - Calls a procedure on a remote machine
  - Idea: Make communication look like an ordinary function call
  - Automate all of the complexity of translating between representations
  - Client calls:
    `remoteFileSystem→Read("rutabaga");`
  - Translated automatically into call on server:
    `fileSys→Read("rutabaga");`

RPC Concept


Client (caller)
r = f(v1, v2);
call
return
bundle
args
Client
Stub
unbundle
ret vals
send
receive
Server (callee)
res_t f(a1, a2)
return
call
bundle
ret vals
Server
Stub
unbundle
args
send
receive

RPC Information Flow



RPC Details (1/3)

- Request-response message passing (under covers!)
- Equivalence with regular procedure call
  - Parameters ⇔ Request Message
  - Result ⇔ Reply message
  - Name of Procedure: Passed in request message
  - Return Address: mbox2 (client return mail box)
- Stub generator: Compiler that generates stubs
  - Input: interface definitions in an “interface definition language (IDL)”
    - » Contains, among other things, types of arguments/return
  - Output: stub code in the appropriate source language
    - » Code for client to pack message, send it off, wait for result, unpack result and return to caller
    - » Code for server to unpack message, call procedure, pack results, send them off

RPC Details (2/3)

- Cross-platform issues:
  - What if client/server machines are different architectures/ languages?
    - » Convert everything to/from some canonical form
    - » Tag every item with an indication of how it is encoded (avoids unnecessary conversions)

- How does client know which mbox (destination queue) to send to?
  - Need to translate name of remote service into network endpoint (Remote machine, port, possibly other info)
  - Binding: the process of converting a user-visible name into a network endpoint
    - » This is another word for "naming" at network level
    - » Static: fixed at compile time
    - » Dynamic: performed at runtime

RPC Details (3/3)

- Dynamic Binding
  - Most RPC systems use dynamic binding via name service
    - » Name service provides dynamic translation of service → mbox
  - Why dynamic binding?
    - » Access control: check who is permitted to access service
    - » Fail-over: If server fails, use a different one

- What if there are multiple servers?
  - Could give flexibility at binding time
    - » Choose unloaded server for each new client
  - Could provide same mbox (router level redirect)
    - » Choose unloaded server for each new request
    - » Only works if no state carried from one call to next

- What if multiple clients?
  - Pass pointer to client-specific return mbox in request

Problems with RPC: Non-Atomic Failures

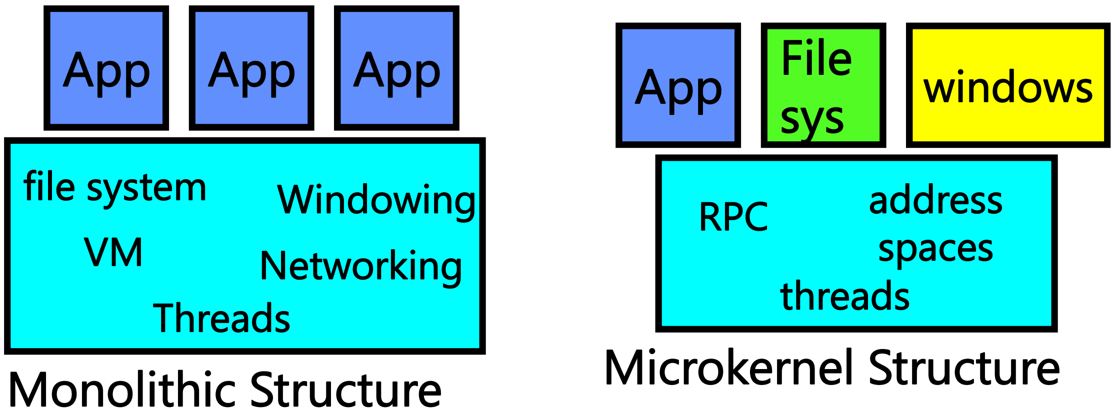
- Different failure modes in dist. system than on a single machine
- Consider many different types of failures
  - User-level bug causes address space to crash
  - Machine failure, kernel bug causes all processes on same machine to fail
  - Some machine is compromised by malicious party
- Before RPC: whole system would crash/die
- After RPC: One machine crashes/compromised while others keep working
- Can easily result in inconsistent view of the world
  - Did my cached data get written back or not?
  - Did server do what I requested or not?
- Answer? Distributed transactions/Byzantine Commit

Cross-Domain Communication/Location Transparency

- How do address spaces communicate with one another?
  - Shared Memory with Semaphores, monitors, etc…
  - File System
  - Pipes (1-way communication)
  - "Remote" procedure call (2-way communication)
- RPC's can be used to communicate between address spaces on different machines or the same machine
  - Services can be run wherever it's most appropriate
  - Access to local and remote services looks the same
- Examples of RPC systems:
  - CORBA (Common Object Request Broker Architecture)
  - DCOM (Distributed COM)
  - RMI (Java Remote Method Invocation)

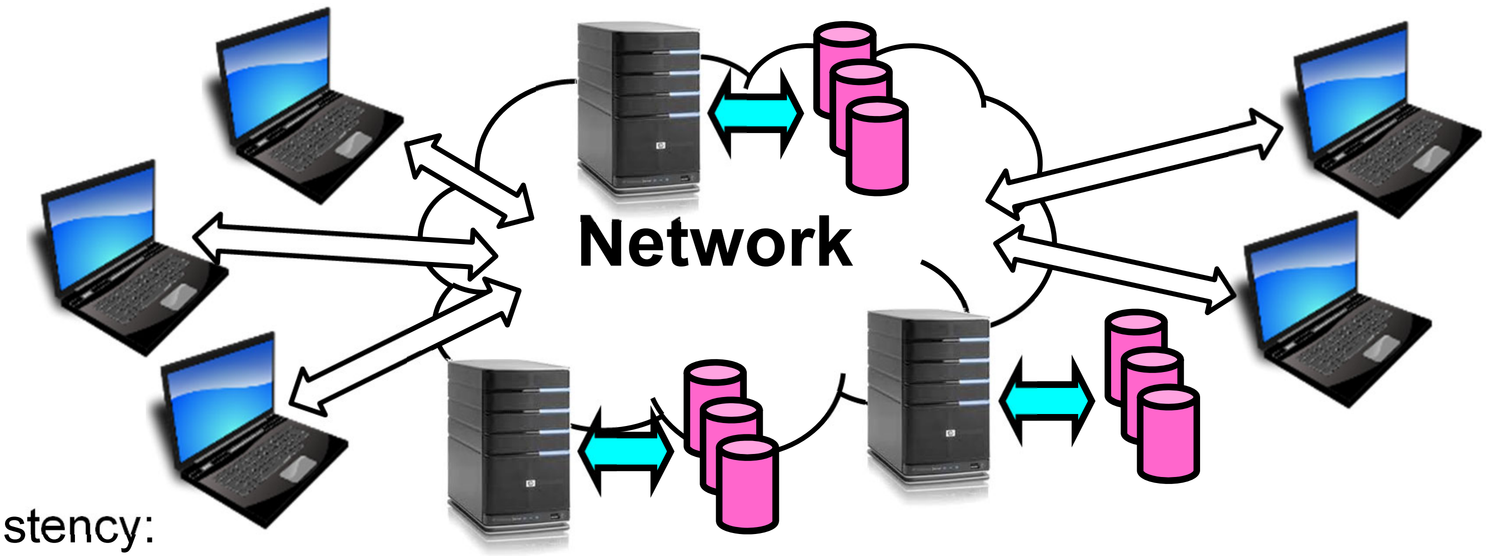
Microkernel operating systems

- Example: split kernel into application-level servers.
  - File system looks remote, even though on same machine



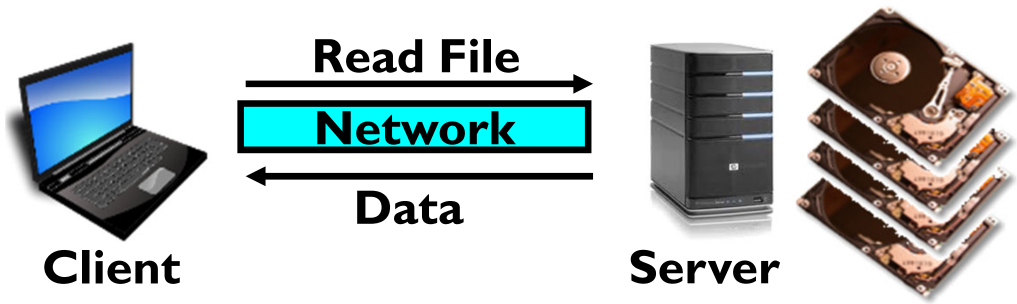
- Why split the OS into separate domains?
  - Fault isolation: bugs are more isolated (build a firewall)
  - Enforces modularity: allows incremental upgrades of pieces of software (client or server)
  - Location transparent: service can be local or remote
    - » For example in the X windowing system: Each X client can be on a separate machine from X server; Neither has to run on the machine with the frame buffer.

Network-Attached Storage and the CAP Theorem

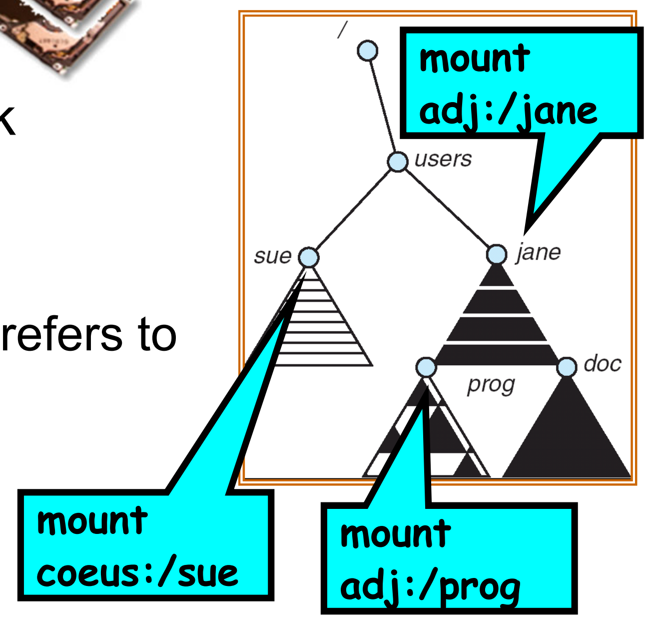


- Consistency:
  - Changes appear to everyone in the same serial order
- Availability:
  - Can get a result at any time
- Partition-Tolerance
  - System continues to work even when network becomes partitioned
- Consistency, Availability, Partition-Tolerance (CAP) Theorem: Cannot have all three at same time
  - Otherwise known as "Brewer's Theorem"

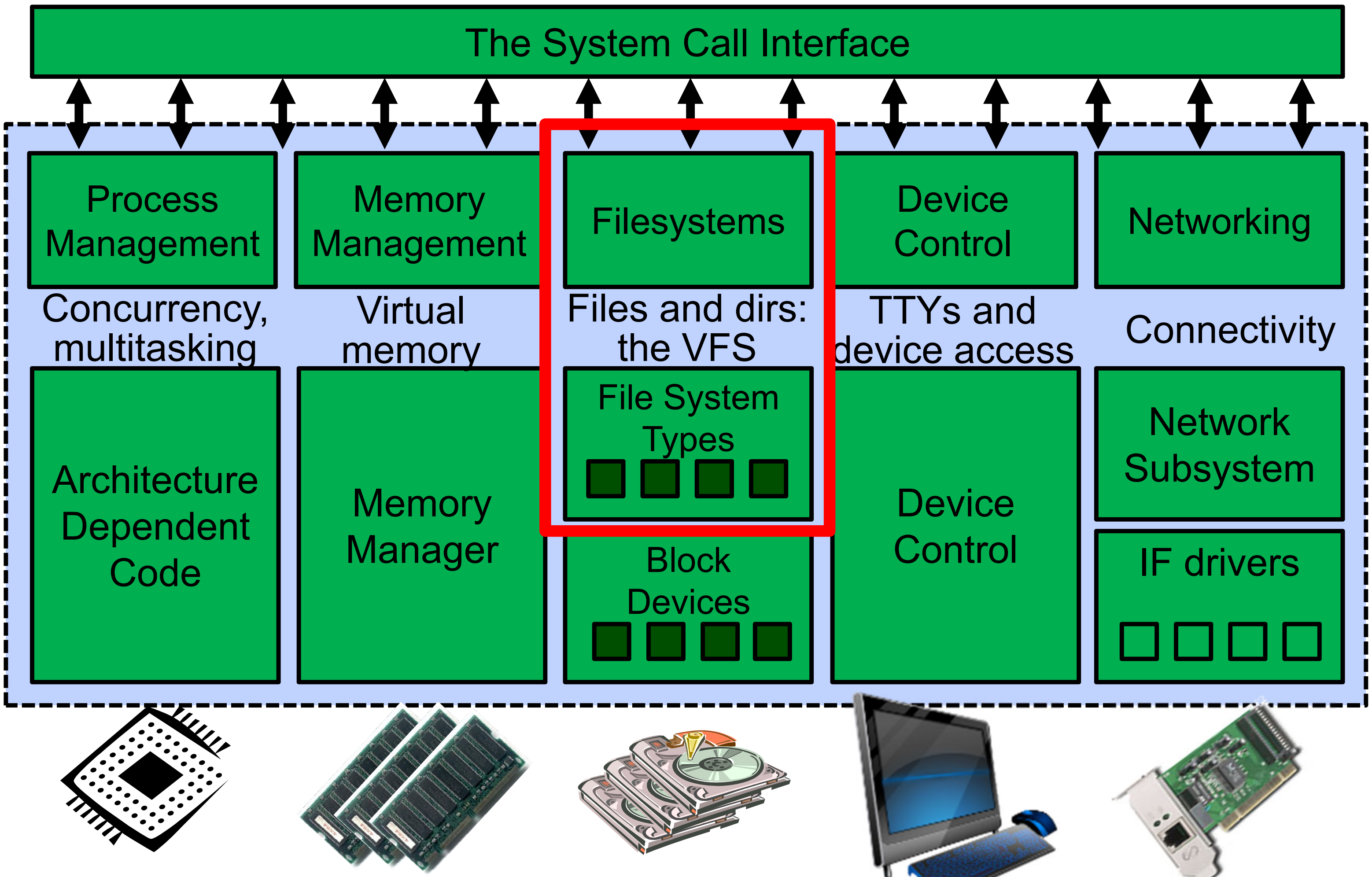
Distributed File Systems



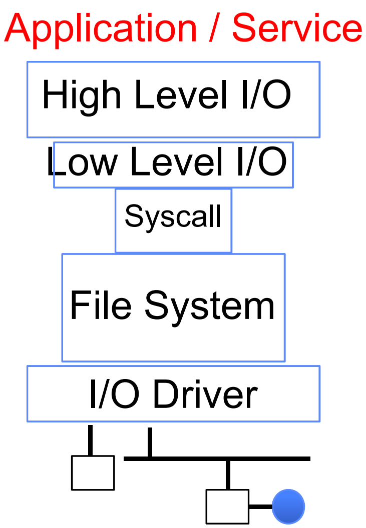
- Transparent access to files stored on a remote disk
- *Mount* remote files into your local file system
  - Directory in local file system refers to remote files
  - e.g., `/users/jane/prog/foo.c` on laptop actually refers to `/prog/foo.c` on `adj.cs.berkeley.edu`
- *Naming* Choices:
  - [Hostname,localname]: Filename includes server
    - » No location or migration transparency, except through DNS remapping
  - A global name space: Filename unique in "world"
    - » Can be served by any server



Enabling Design: VFS



Recall: Layers of I/O…



User App:

```
length = read(input_fd, buffer, BUFFER_SIZE);
```

User library:

```
ssize_t read(int, void *, size_t) {
  marshal args into registers
  issue syscall
  register result of syscall to rtn value
};
```

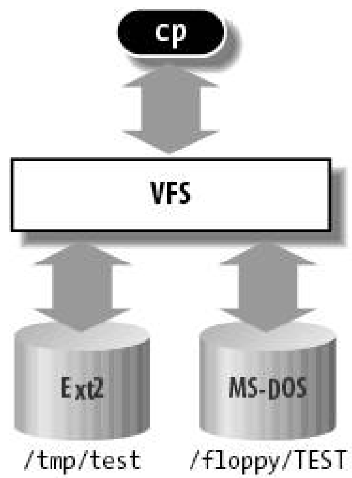
Exception U→K, interrupt processing

```
void syscall_handler (struct intr_frame *f) {
   unmarshall call#, args from regs
   dispatch : handlers[call#](args)
   marshal results fo syscall ret
}
```

```
ssize_t vfs_read(struct file *file, char __user *buf,
                 size_t count, loff_t *pos) {
   User Process/File System relationship
   call device driver to do the work
}
```

Device Driver

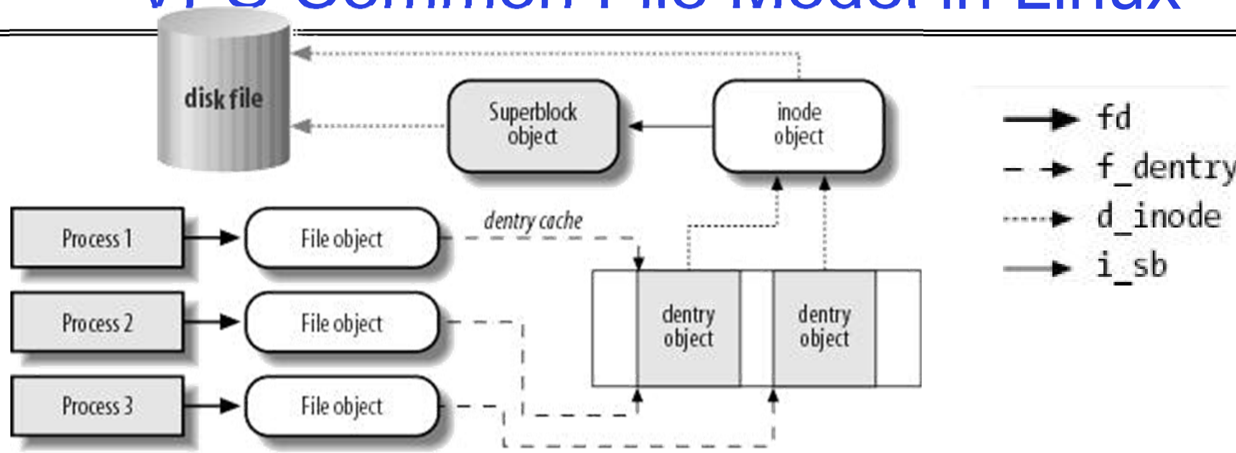
Virtual Filesystem Switch




```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
       O_WRONLY|O_CREAT|O_TRUNC, 0600);
do {
    i = read(inf, buf, 4096);
    write(outf, buf, i);
} while (i);
close(outf);
close(inf);
```

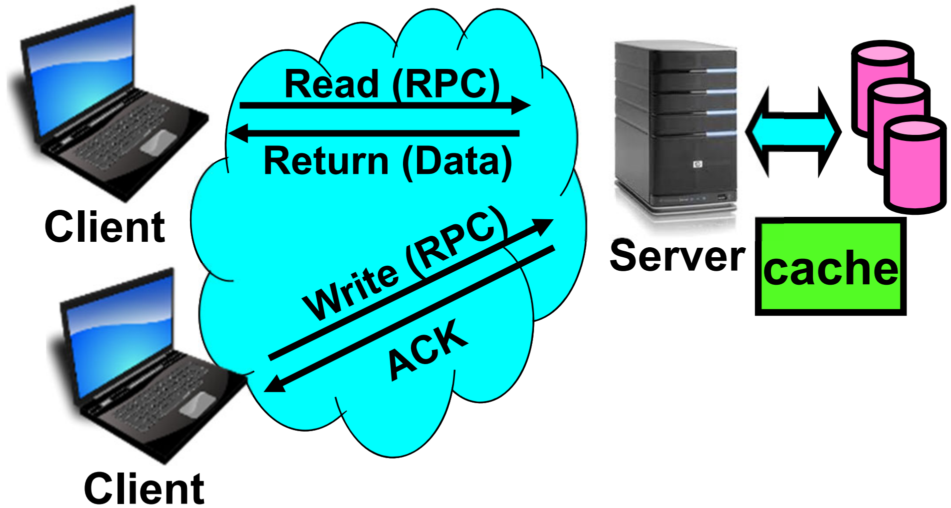
- VFS: Virtual abstraction similar to local file system
  - Provides virtual superblocks, inodes, files, etc
  - Compatible with a variety of local and remote file systems
    - » provides object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - The API is to the VFS interface, rather than any specific type of file system

VFS Common File Model in Linux



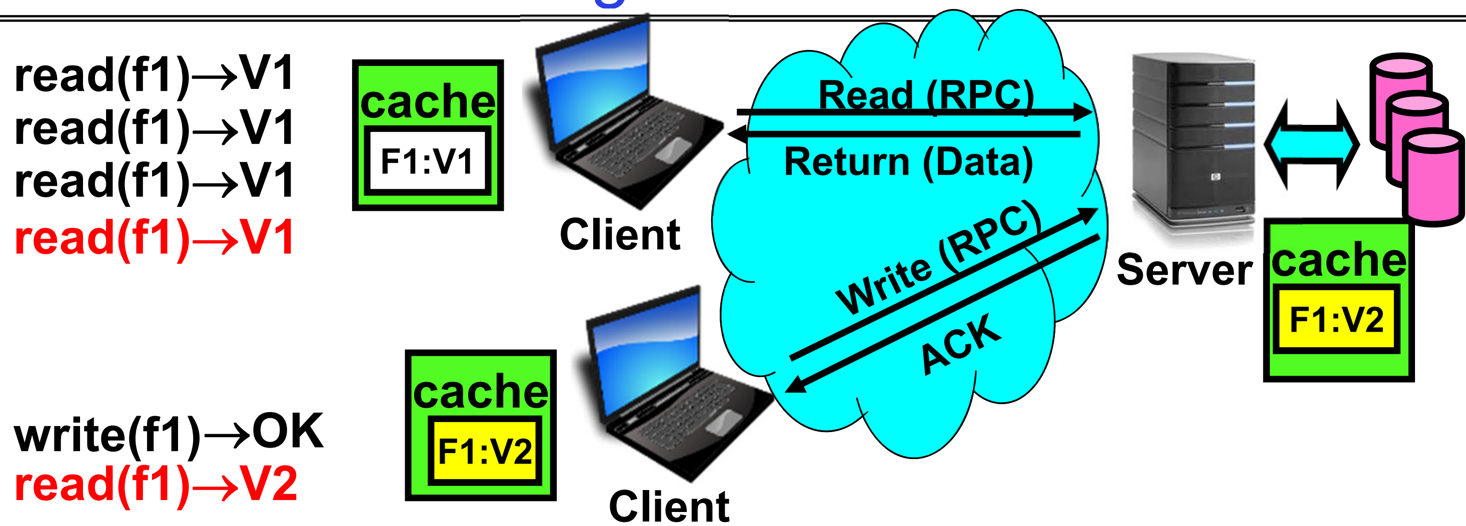
- Four primary object types for VFS:
  - superblock object: represents a specific mounted filesystem
  - inode object: represents a specific file
  - dentry object: represents a directory entry
  - file object: represents open file associated with process
- There is no specific directory object (VFS treats directories as files)
- May need to fit the model by faking it
  - Example: make it look like directories are files
  - Example: make it look like have inodes, superblocks, etc.

Simple Distributed File System



- Remote Disk: Reads and writes forwarded to server
  - Use Remote Procedure Calls (RPC) to translate file system calls into remote requests
  - No local caching, but can be cache at server-side
- Advantage: Server provides consistent view of file system to multiple clients
- Problems? Performance!
  - Going over network is slower than going to local memory
  - Lots of network traffic/not well pipelined
  - Server can be a bottleneck

Use of caching to reduce network load



- Idea: Use caching to reduce network load
  - In practice: use buffer cache at source and destination
- Advantage: if open/read/write/close can be done locally, don't need to do any network traffic…fast!
- Problems:
  - Failure:
    - » Client caches have data not committed at server
  - Cache consistency!
    - » Client caches not consistent with server/each other

Dealing with Failures

- What if server crashes? Can client wait until it comes back and just continue making requests?
  – Changes in server's cache but not in disk are lost
- What if there is shared state across RPC's?
  – Client opens file, then does a seek
  – Server crashes
  – What if client wants to do another read?
- Similar problem: What if client removes a file but server crashes before acknowledgement?

Stateless Protocol

- Stateless Protocol: A protocol in which all information required to service a request is included with the request
- Even better: Idempotent Operations – repeating an operation multiple times is same as executing it just once (e.g., storing to a mem addr.)
- Client: timeout expires without reply, just run the operation again (safe regardless of first attempt)

- Recall HTTP: Also a stateless protocol
  - Include cookies with request to simulate a session

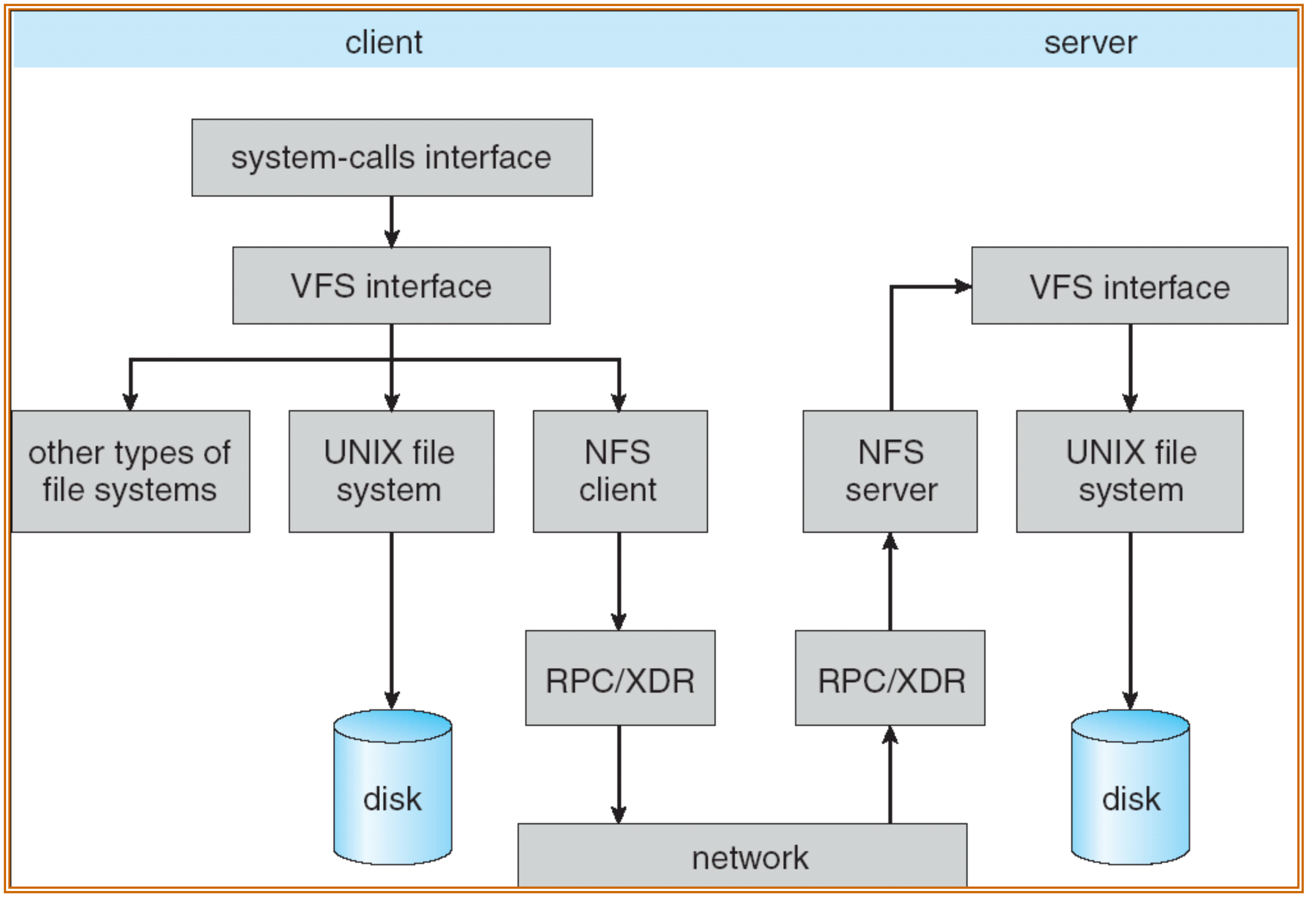
Case Study: Network File System (NFS)

- Three Layers for NFS system
  - UNIX file-system interface: open, read, write, close calls + file descriptors
  - VFS layer: distinguishes local from remote files
    - » Calls the NFS protocol procedures for remote requests
  - NFS service layer: bottom layer of the architecture
    - » Implements the NFS protocol
- NFS Protocol: RPC for file operations on server
  - XDR Serialization standard for data format independence
  - Reading/searching a directory
  - manipulating links and directories
  - accessing file attributes/reading and writing files
- Write-through caching: Modified data committed to server's disk before results are returned to the client
  - lose some of the advantages of caching
  - time to perform write() can be long
  - Need some mechanism for readers to eventually notice changes! (more on this later)

NFS Continued

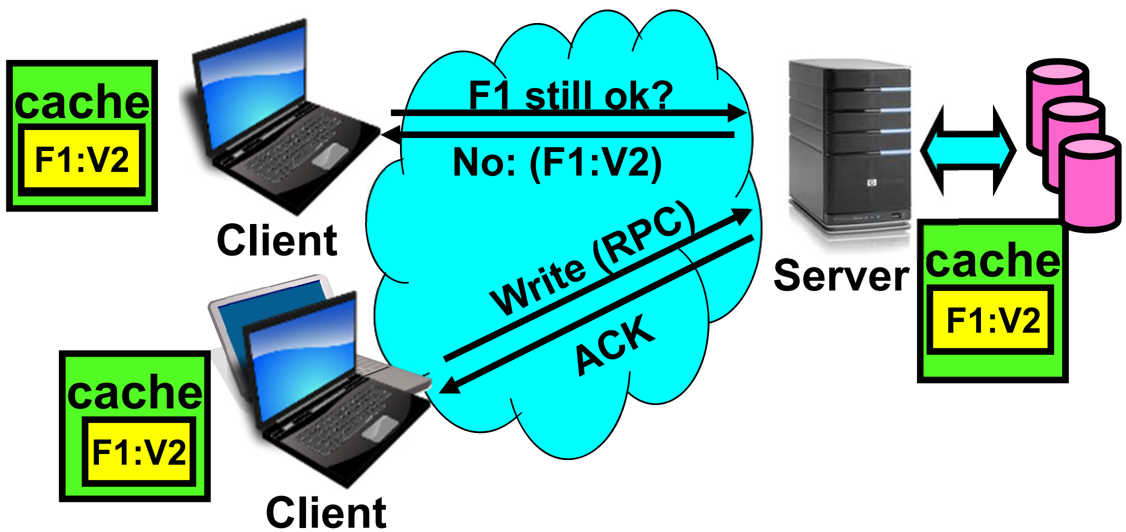
- NFS servers are stateless; each request provides all arguments require for execution
  - E.g. reads include information for entire operation, such as **`ReadAt(inumber,position)`**, not **`Read(openfile)`**
  - No need to perform network open() or close() on file – each operation stands on its own
- Idempotent: Performing requests multiple times has same effect as performing them exactly once
  - Example: Server crashes between disk I/O and message send, client resend read, server does operation again
  - Example: Read and write file blocks: just re-read or re-write file block – no other side effects
  - Example: What about "remove"? NFS does operation twice and second time returns an advisory error
- Failure Model: Transparent to client system
  - Is this a good idea? What if you are in the middle of reading a file and server crashes?
  - Options (NFS Provides both):
    - » Hang until server comes back up (next week?)
    - » Return an error. (Of course, most applications don't know they are talking over network)

NFS Architecture



NFS Cache consistency

- NFS protocol: weak consistency
  - Client polls server periodically to check for changes
    - » Polls server if data hasn't been checked in last 3-30 seconds (exact timeout it tunable parameter).
    - » Thus, when file is changed on one client, server is notified, but other clients use old version of file until timeout.



  - What if multiple clients write to same file?
    - » In NFS, can get either version (or parts of both)
    - » Completely arbitrary!

What about: Sharing Data, rather than Files ?

- Key:Value stores are used everywhere
- Native in many programming languages
  - Associative Arrays in Perl
  - Dictionaries in Python
  - Maps in Go
  - …
- What about a collaborative key-value store rather than message passing or file sharing?
- Can we make it scalable and reliable?

Key Value Storage

Simple interface

- **put(key, value);** `// Insert/write "value" associated with key`

- **get(key);** `// Retrieve/read value associated with key`

Why Key Value Storage?

- Easy to Scale
  - Handle huge volumes of data (e.g., petabytes)
  - Uniform items: distribute easily and roughly equally across many machines

- Simple consistency properties

- Used as a simpler but more scalable "database"
  - Or as a building block for a more capable DB

Key Values: Examples

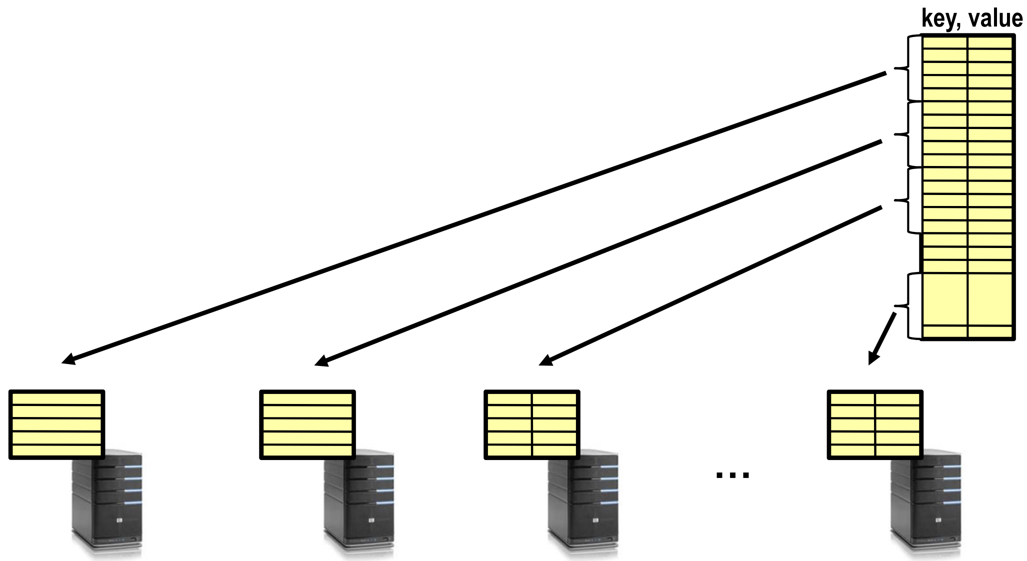
- Amazon:
  - Key: customerID
  - Value: customer profile (e.g … credit card, ..)
- Facebook, Twitter:
  - Key: UserID
  - Value: user profile (e.g., posting history, photos, friends, …)
- iCloud/iTunes:
  - Key: Movie/song name
  - Value: Movie, Song

Key-value storage systems in real life

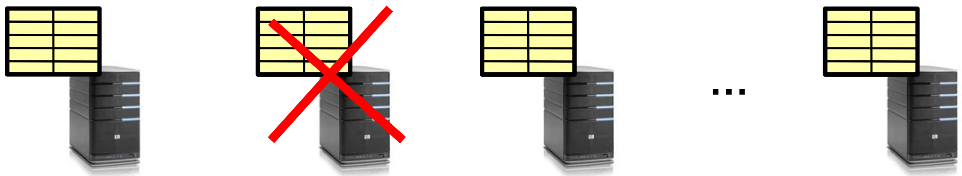
- **Amazon**
  - DynamoDB: internal key value store used to power Amazon.com (shopping cart)
  - Simple Storage System (S3)
- **BigTable/HBase/Hypertable:** distributed, scalable data storage
- **Cassandra**: “distributed data management system” (developed by Facebook)
- **Memcached:** in-memory key-value store for small chunks of arbitrary data (strings, objects)
- **eDonkey/eMule:** peer-to-peer sharing system
- …

Key Value Store

- Also called Distributed Hash Tables (DHT)
- Main idea: simplify storage interface (i.e. put/get), then partition set of key-values across many machines



Challenges



- **Scalability:**
  - Need to scale to thousands of machines
  - Need to allow easy addition of new machines
- **Fault Tolerance:** handle machine failures without losing data and without degradation in performance
- **Consistency:** maintain data consistency in face of node failures and message losses
- **Heterogeneity** (if deployed as peer-to-peer systems):
  - Latency: 1ms to 1000ms
  - Bandwidth: 32Kb/s to 100Mb/s

Important Questions

- **put(key, value)**:
  – where do you store a new (key, value) tuple?
- **get(key)**:
  – where is the value associated with a given “key” stored?

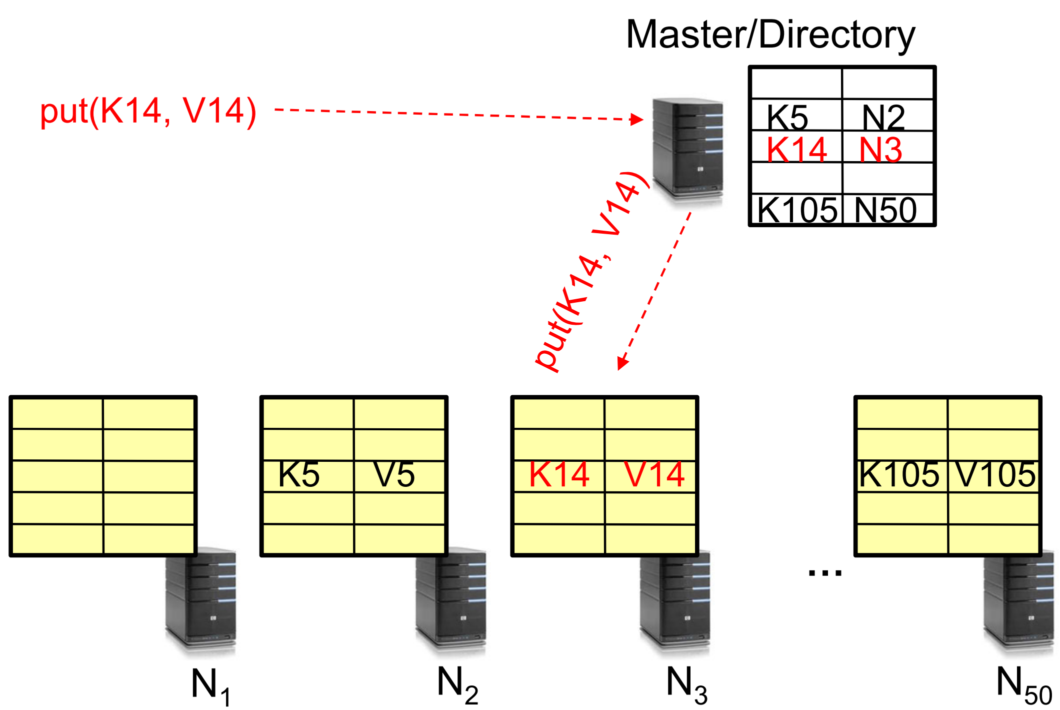
- And, do the above while providing
  – Scalability
  – Fault Tolerance
  – Consistency

How to solve the "where?"

- Hashing to map key space ⇒ location
  - But what if you don't know all the nodes that are participating?
  - Perhaps they come and go …
  - What if some keys are really popular?
- Lookup
  - Hmm, won't this be a bottleneck and single point of failure?

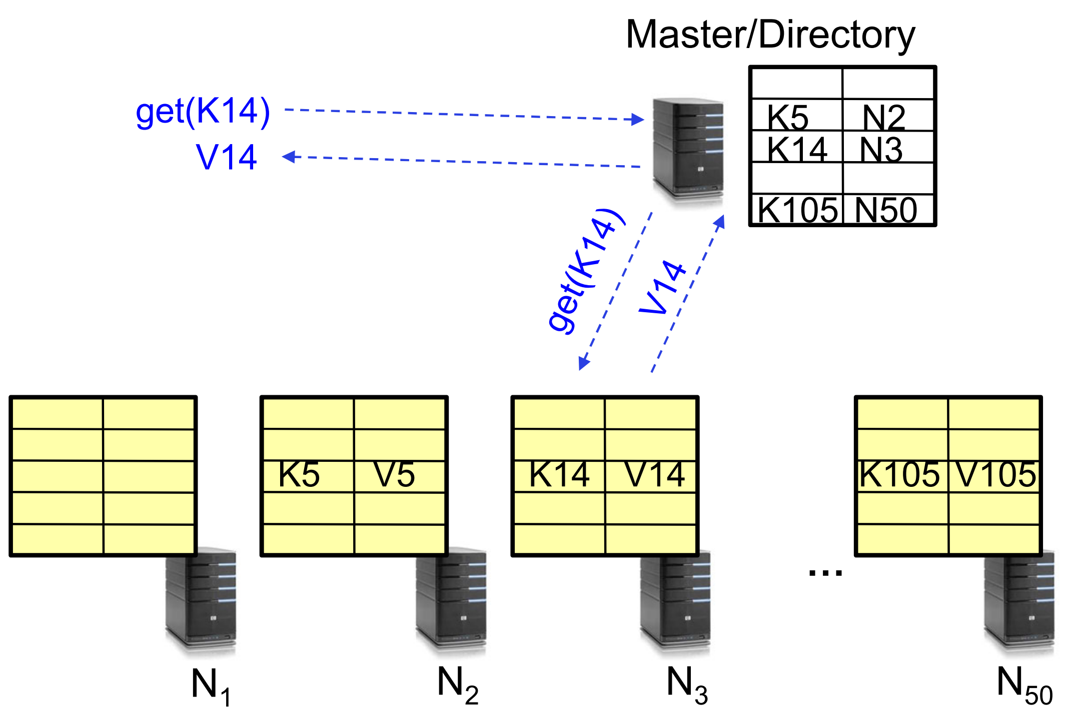
Recursive Directory Architecture (put)

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**



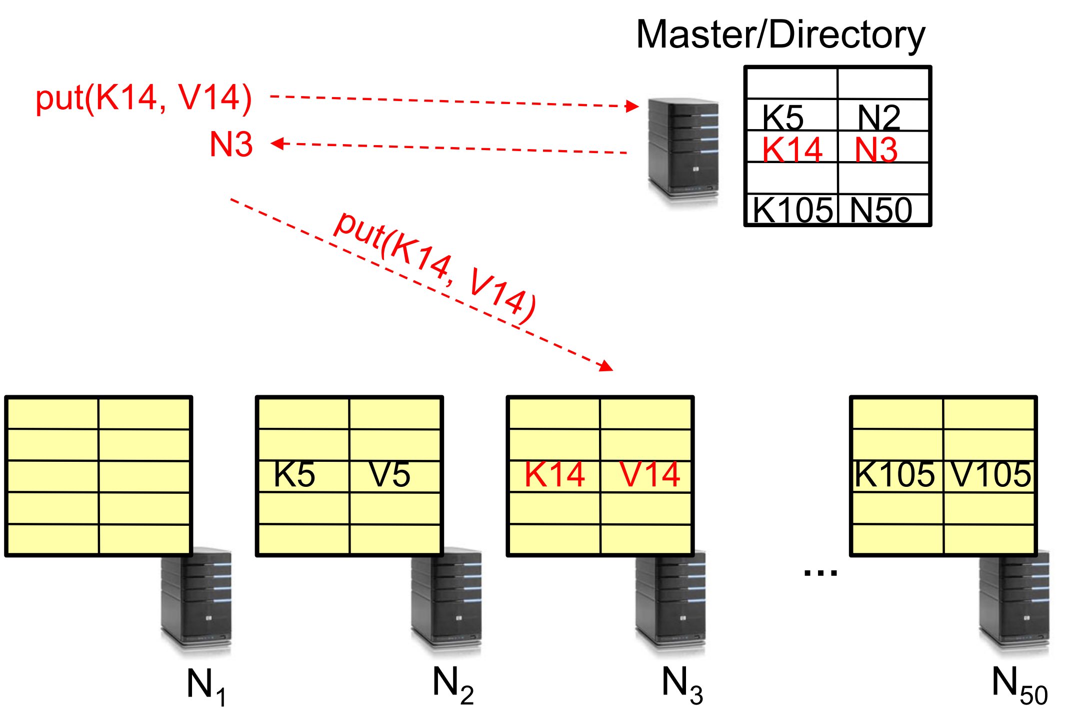
Recursive Directory Architecture (get)

- Have a node maintain the mapping between **keys** and the **machines (nodes)** that store the **values** associated with the **keys**



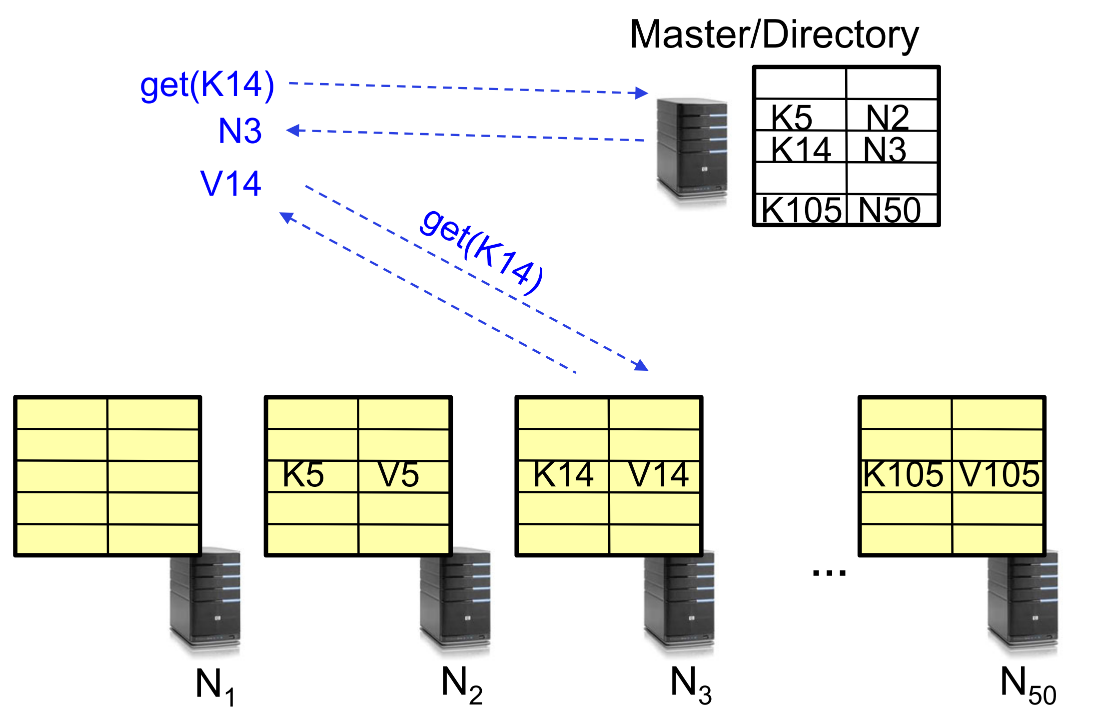
Iterative Directory Architecture (put)

- Having the master relay the requests → **recursive query**
- Another method: **iterative query** (this slide)
  - – Return node to requester and let requester contact node

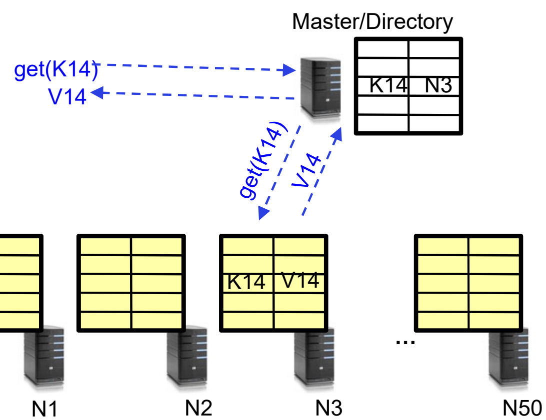


Iterative Directory Architecture (get)

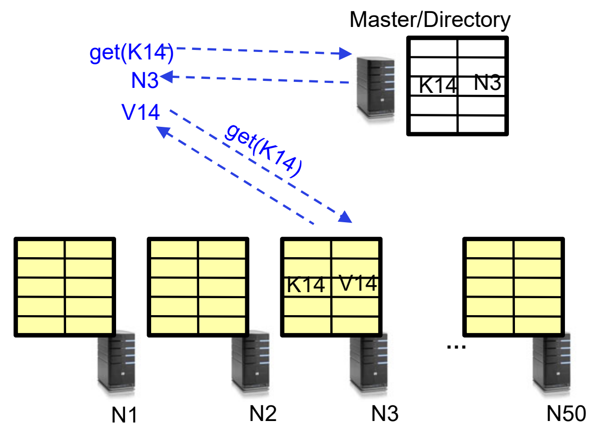
- Having the master relay the requests → **recursive query**
- Another method: **iterative query** (this slide)
  - – Return node to requester and let requester contact node



Iterative vs. Recursive Query



Recursive



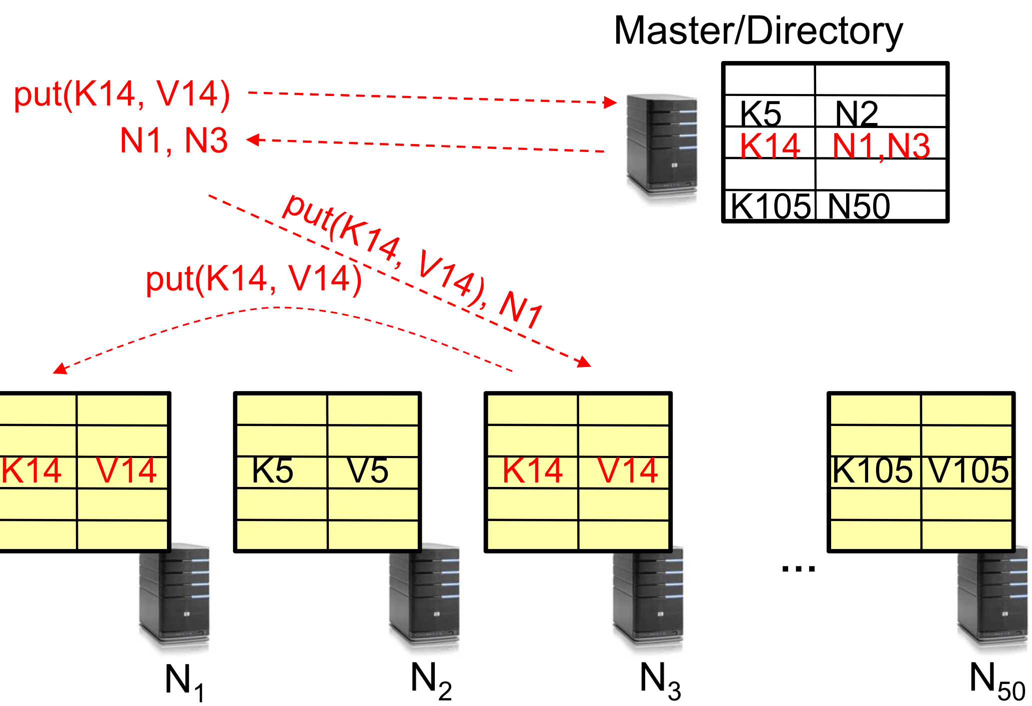
Iterative

- \+ Faster, as directory server is typically close to storage nodes
- \+ Easier for consistency: directory can enforce an order for all puts and gets
- \- Directory is a performance bottleneck

- \+ More scalable, clients do more work
- \- Harder to enforce consistency

Fault Tolerance

- Replicate value on several nodes
- Usually, place replicas on different racks in a datacenter to guard against rack failures



Scalability

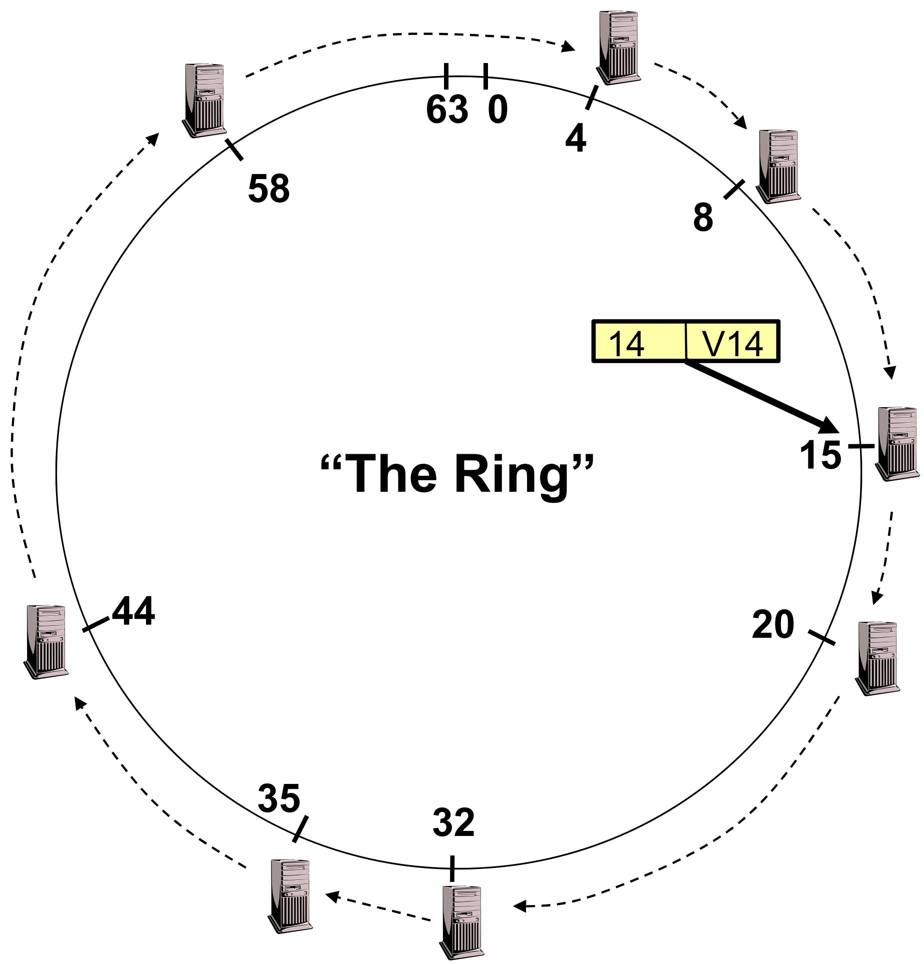
- Storage: use more nodes
- Number of requests:
  - Can serve requests from all nodes on which a value is stored in parallel
  - Master can replicate a popular value on more nodes
- Master/directory scalability:
  - Replicate it
  - Partition it, so different keys are served by different masters/directories
    - » How do you partition?

Scaling Up Directory

- Challenge:
  - Directory contains a number of entries equal to number of (key, value) tuples in the system
  - Can be tens or hundreds of billions of entries in the system!
- Solution: **Consistent Hashing**
  - **Provides mechanism to divide [key,value] pairs amongst a (potentially large!) set of machines (nodes) on network**
- Associate to each node a unique *id* in an *uni*-dimensional space $0..2^m-1$
  $\Rightarrow$ Wraps around: Call this “the ring!”
  - Partition this space across *n* machines
  - Assume keys are in same uni-dimensional space
  - Each [Key, Value] is stored at the node with the smallest ID larger than Key

Key to Node Mapping Example

- Paritioning example with m = 6 → ID space: 0..63
  - Node 8 maps keys [5,8]
  - Node 15 maps keys [9,15]
  - Node 20 maps keys [16, 20]
  - …
  - Node 4 maps keys [59, 4]
- For this example, the mapping [14, V14] maps to node with ID=15
  - Node with smallest ID larger than 14 (the key)
- In practice, m=256 or more!
  - Uses cryptographically secure hash such as SHA-256 to generate the node IDs

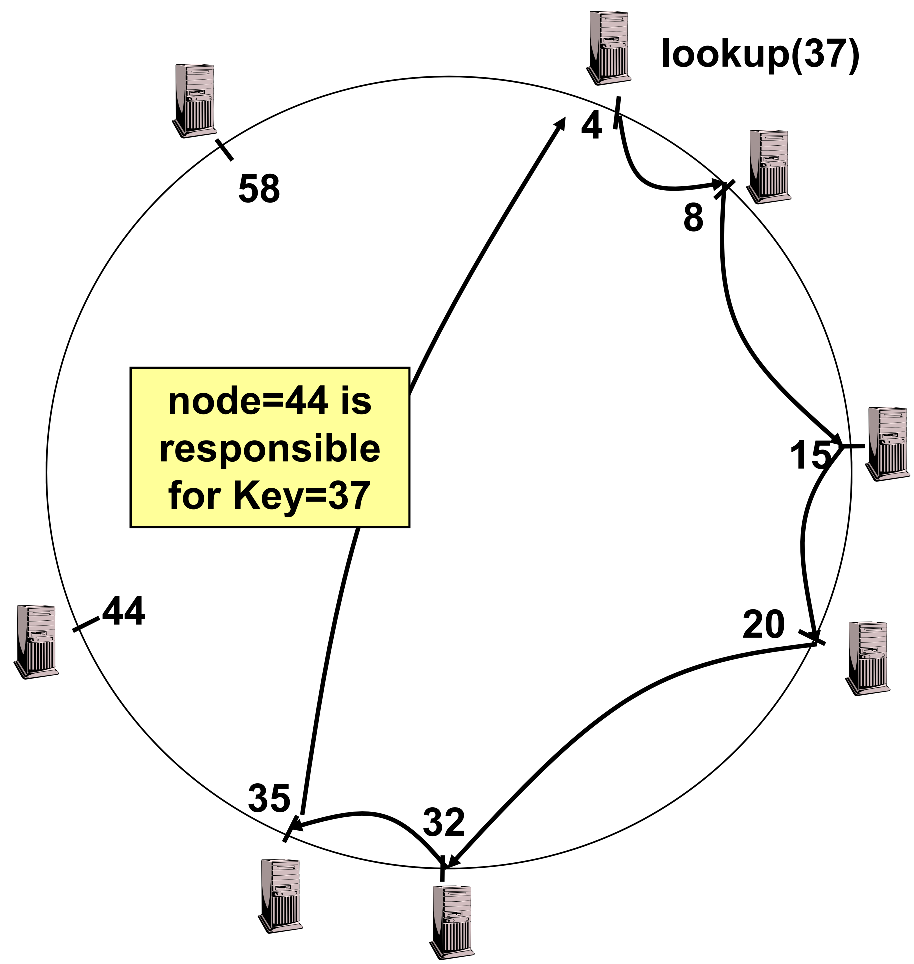


Chord: Distributed Lookup (Directory) Service

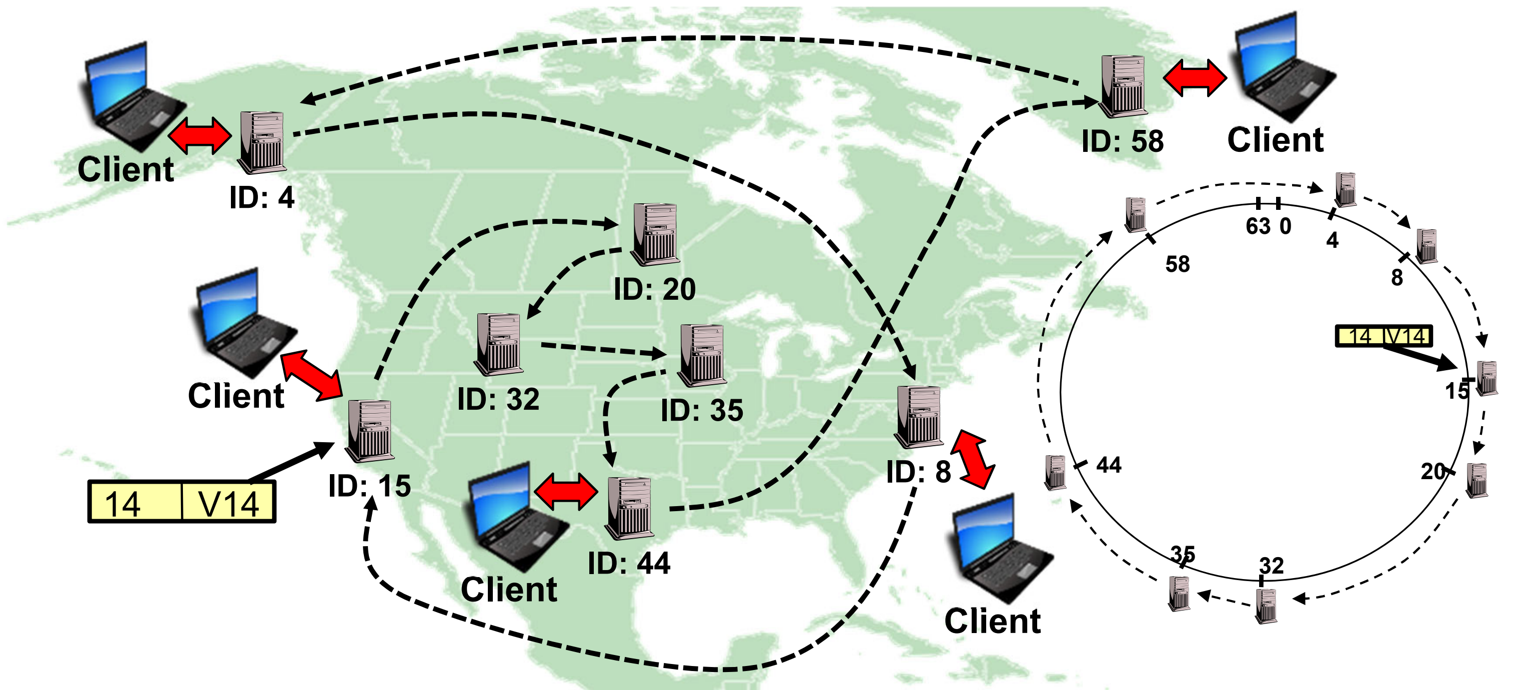
- “Chord” is a Distributed Lookup Service
  - Designed at MIT and here at Berkeley (Ion Stoica among others)
  - Simplest and cleanest algorithm for distributed storage
    - » Serves as comparison point for other optims
- Import aspect of the design space:
  - Decouple correctness from efficiency
  - Combined *Directory* and *Storage*
- Properties
  - Correctness:
    - » Each node needs to know about neighbors on ring (one predecessor and one successor)
    - » Connected rings will perform their task correctly
  - Performance:
    - » Each node needs to know about O(log(M)), where M is the total number of nodes
    - » Guarantees that a tuple is found in O(log(M)) steps
- Many other *Structured, Peer-to-Peer* lookup services:
  - CAN, Tapestry, Pastry, Bamboo, Kademlia, …
  - Several designed here at Berkeley!

Chord's Lookup Mechanism: Routing!

- Each node maintains pointer to its successor
- Route packet (Key, Value) to the node responsible for ID using successor pointers
  - E.g., node=4 lookups for node responsible for Key=37
- Worst-case (correct) lookup is O(n)
  - But much better normal lookup time is O(log n)
  - Dynamic performance optimization (finger table mechanism)
    - » More later!!!



But what does this really mean??



- Node names intentionally scrambled WRT geography!
  - Node IDs generated by secure hashes over metadata
    - » Including things like the IP address
  - This geographic scrambling spreads load and avoids hotspots
- Clients access distributed storage through any member of the network

Stabilization Procedure

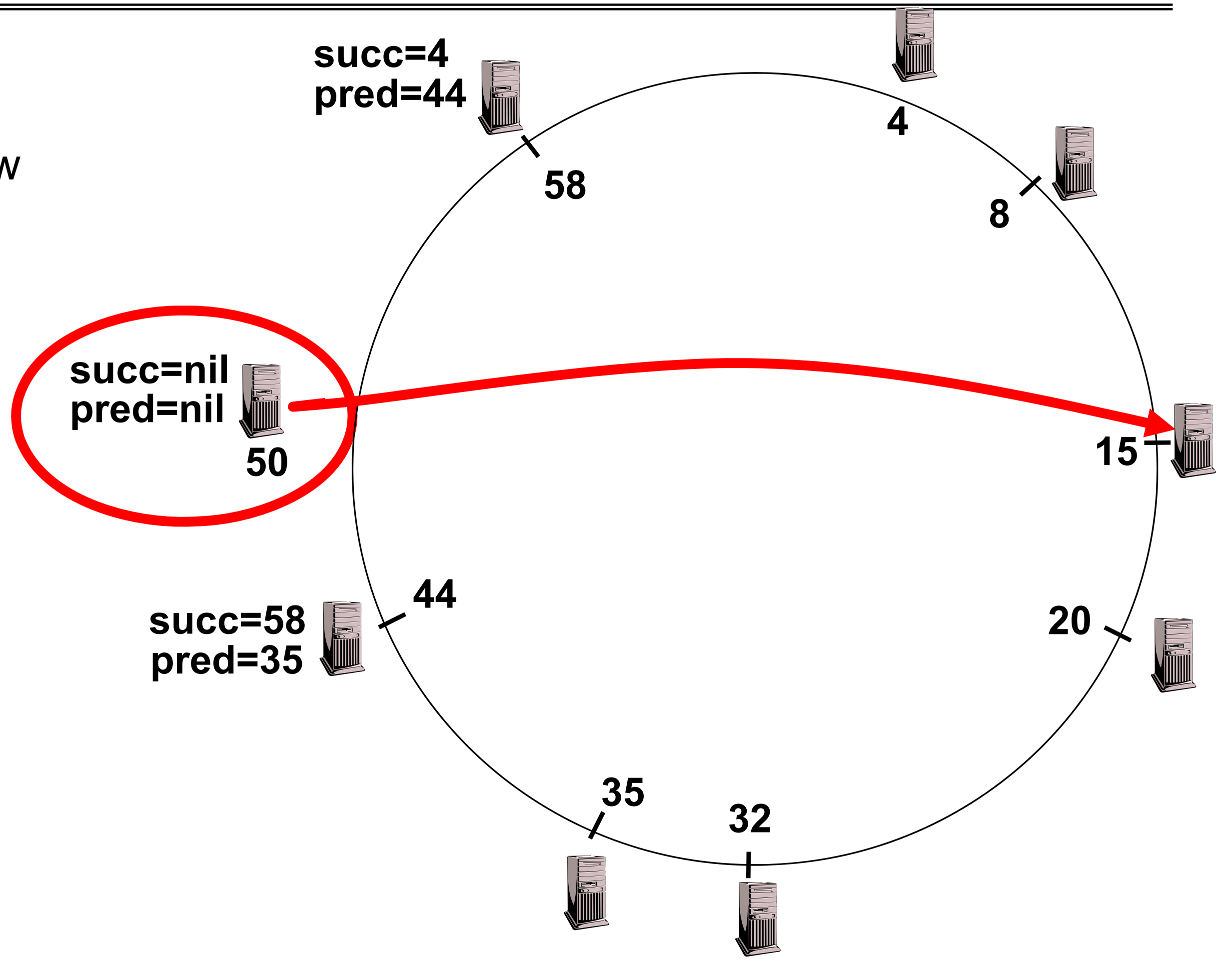
- Periodic operation performed by each node n to maintain its successor when new nodes join the system
  - The primary Correctness constraint

```
n.stabilize()
   x = succ.pred;
   if (x ∈ (n, succ))
      succ = x;       // if x better successor, update
   succ.notify(n); // n tells successor about itself

n.notify(n')
   if (pred = nil or n' ∈ (pred, n))
      pred = n';       // if n' is better predecessor, update
```

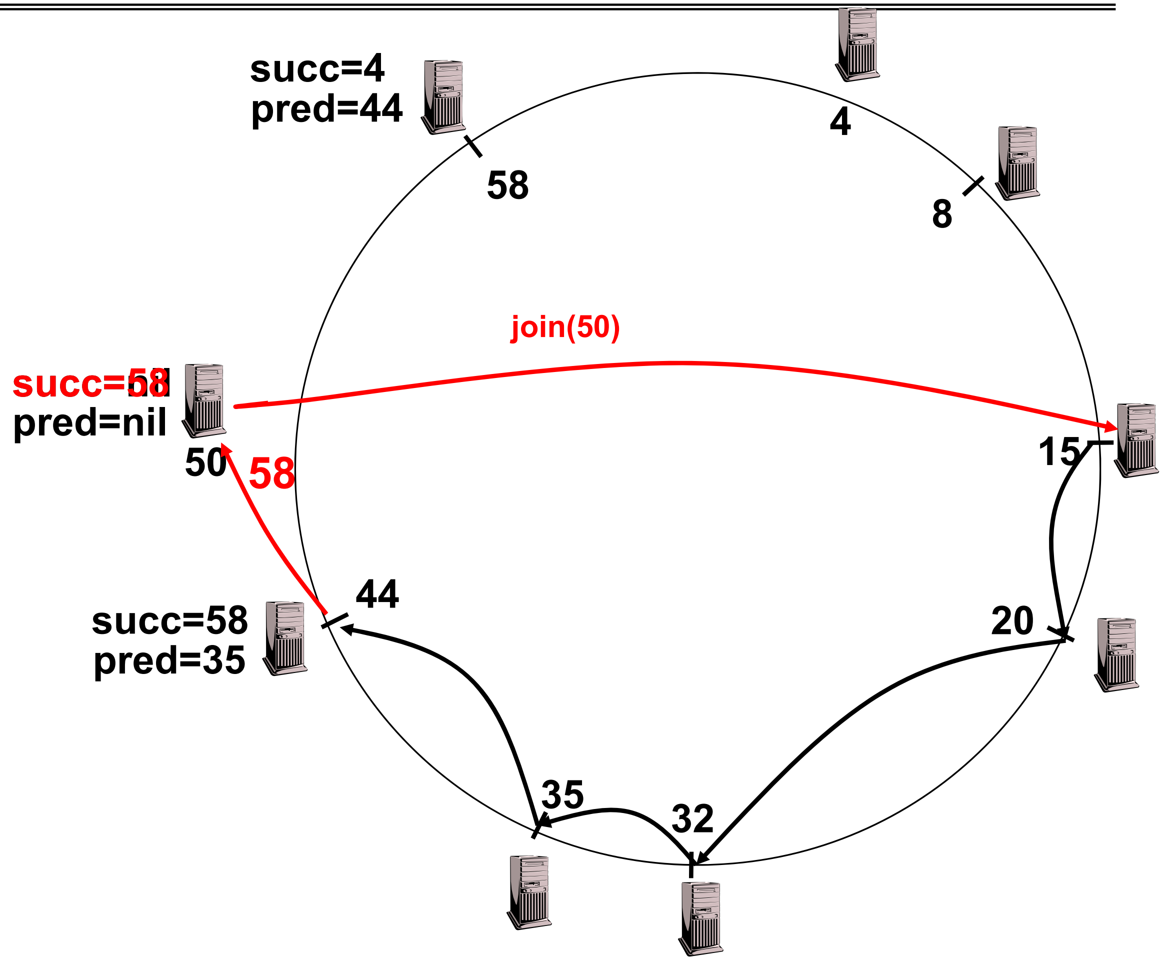
Joining Operation

- Node with id=50 joins the ring
- Node 50 must know at least one node already in system
  - Assume known node is 15



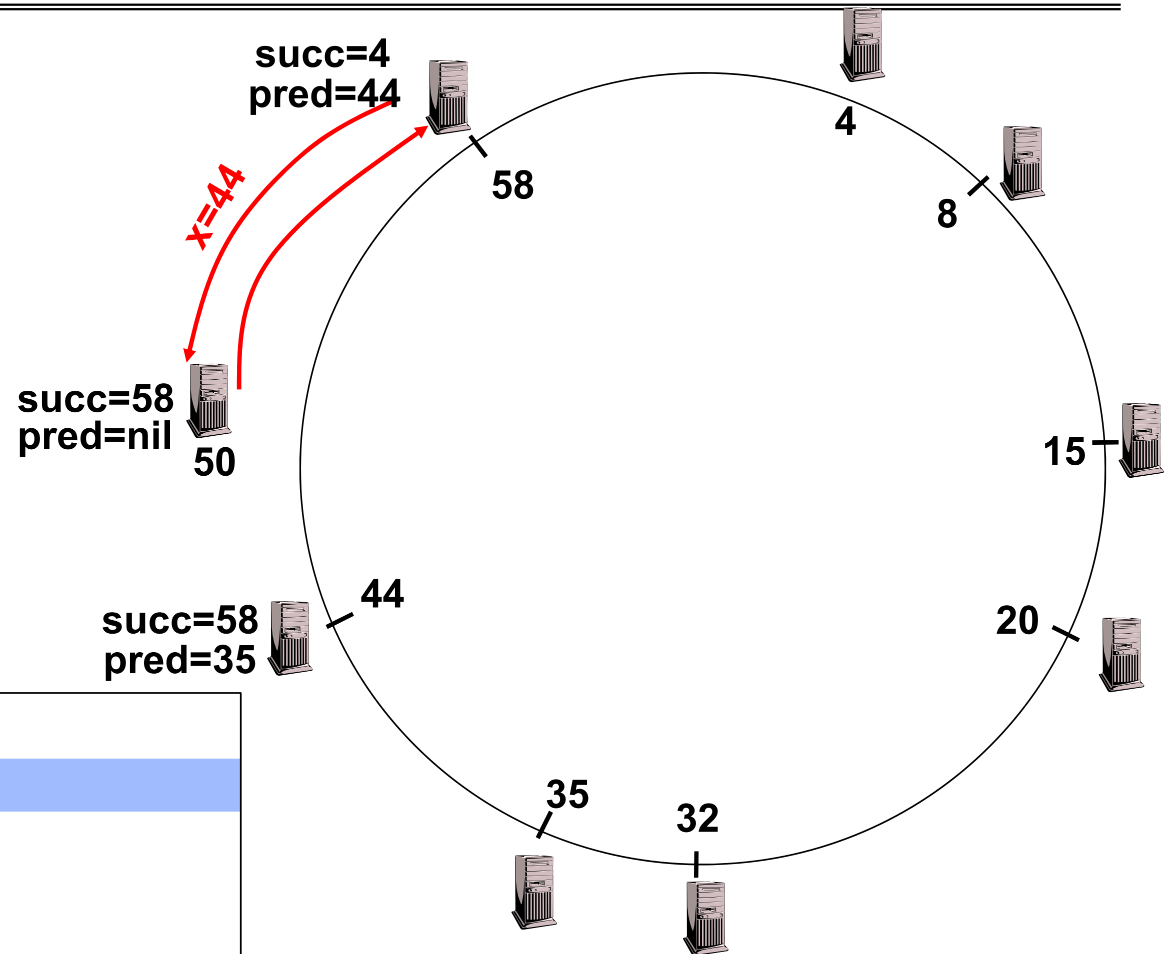
Joining Operation

- n=50 sends join(50) to node 15
  - Join propagated around ring!
- n=44 returns node 58
- n=50 updates its successor to 58



Joining Operation

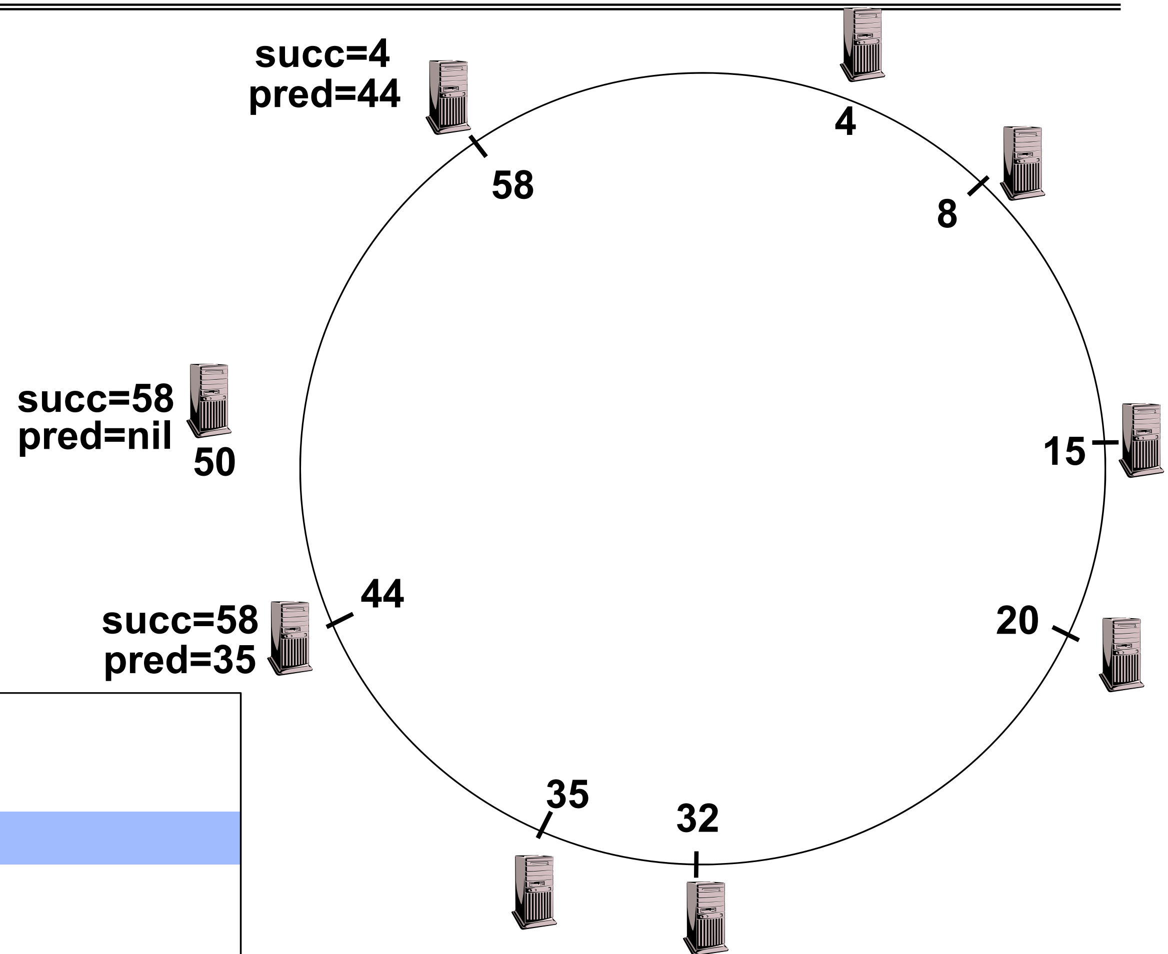
- n=50 executes stabilize()
- n's successor (58) returns x = 44




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
    succ = x;
  succ.notify(n);
```

Joining Operation

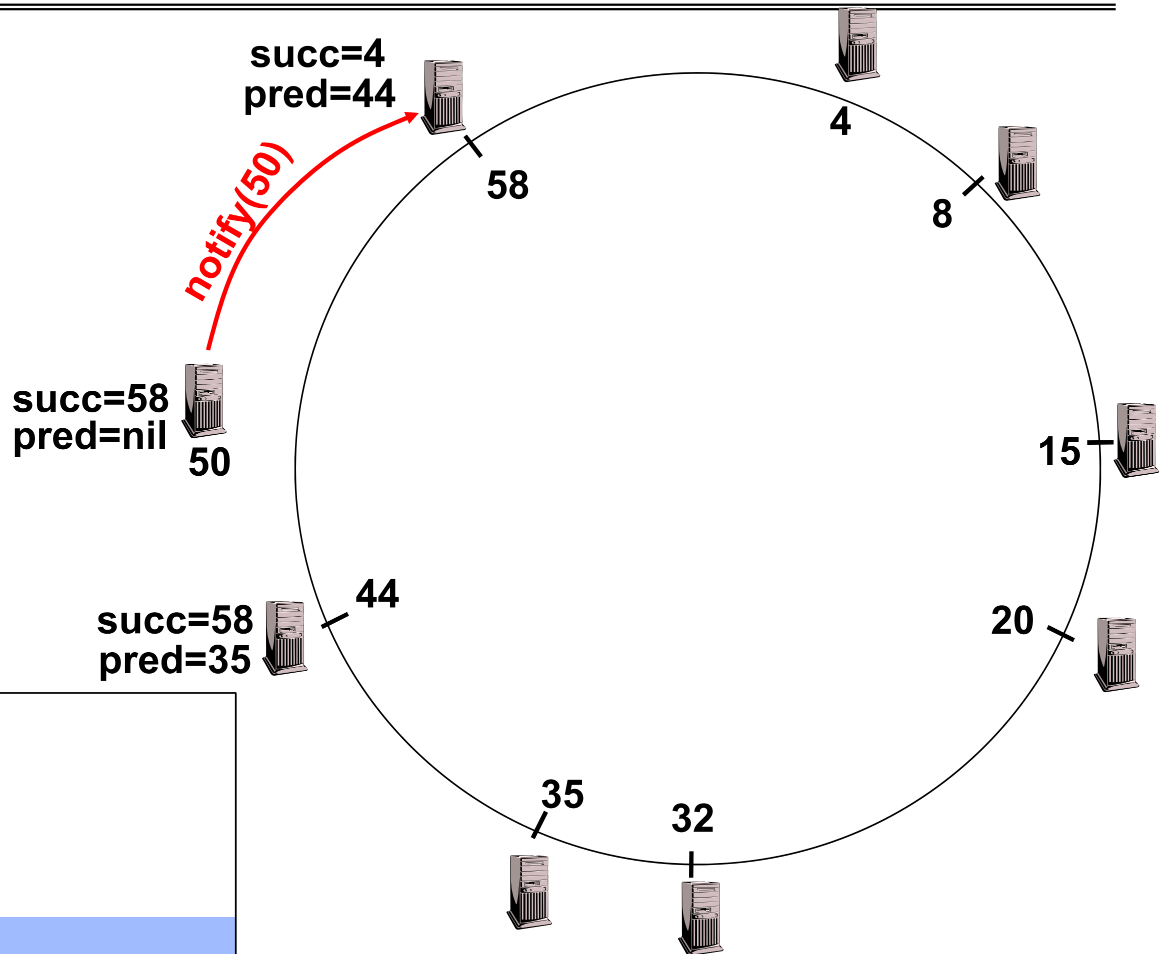
- n=50 executes stabilize()
  - x = 44
  - succ = 58




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
     succ = x;
  succ.notify(n);
```

Joining Operation

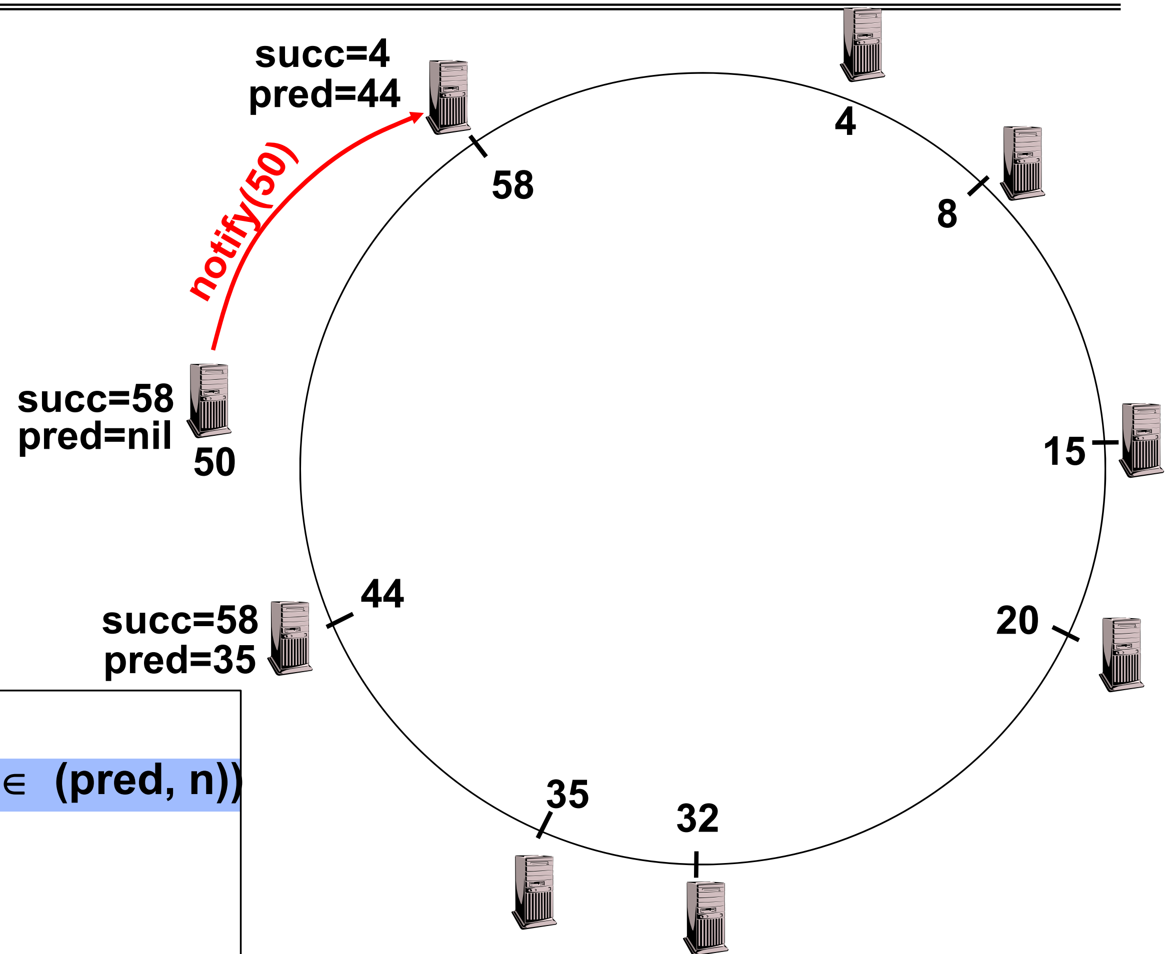
- n=50 executes stabilize()
  - x = 44
  - succ = 58
- n=50 sends to it's successor (58) notify(50)




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
     succ = x;
  succ.notify(n);
```

Joining Operation

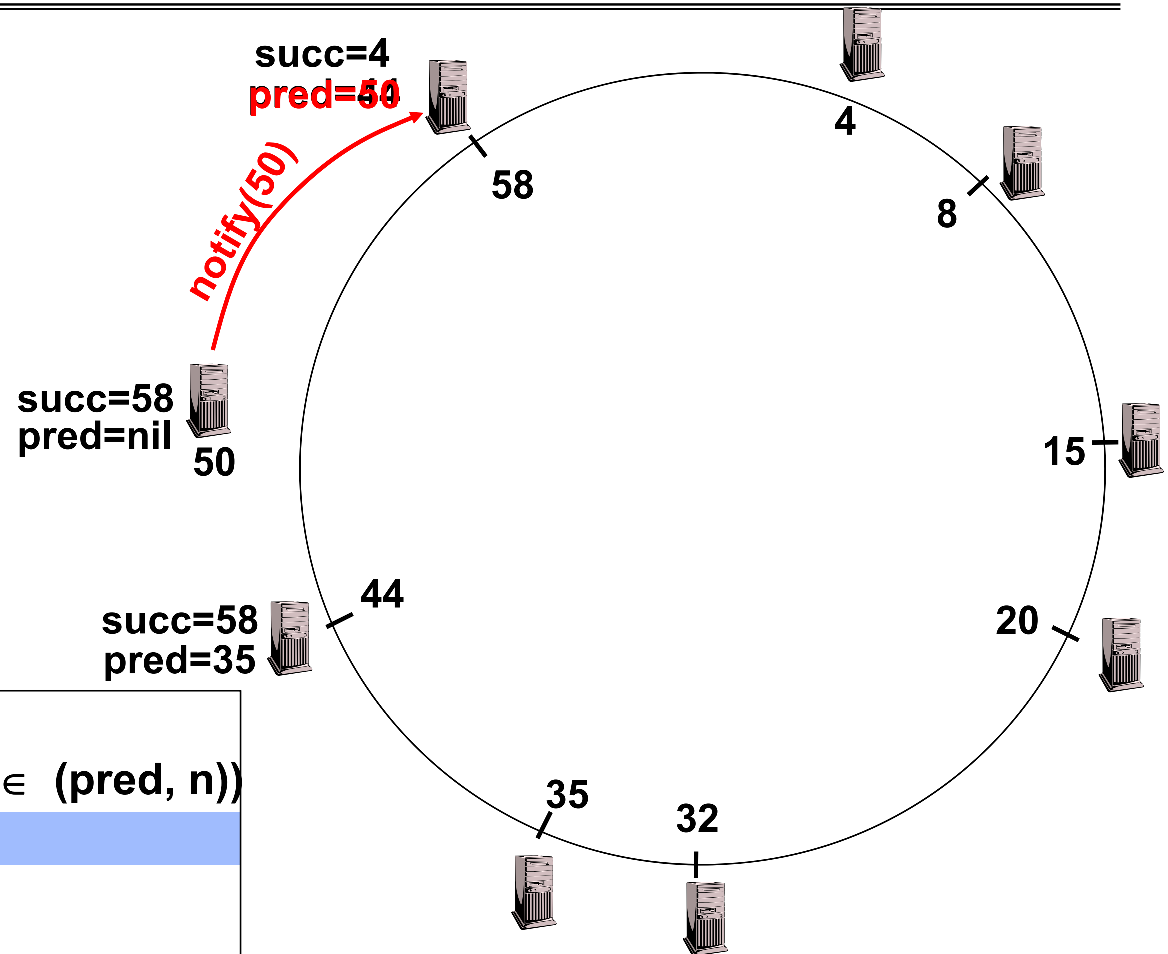
- n=58 executes notify(50)
  - pred = 44
  - n' = 50



n.notify(n')
if (pred = nil or n' ∈ (pred, n))
pred = n'

Joining Operation

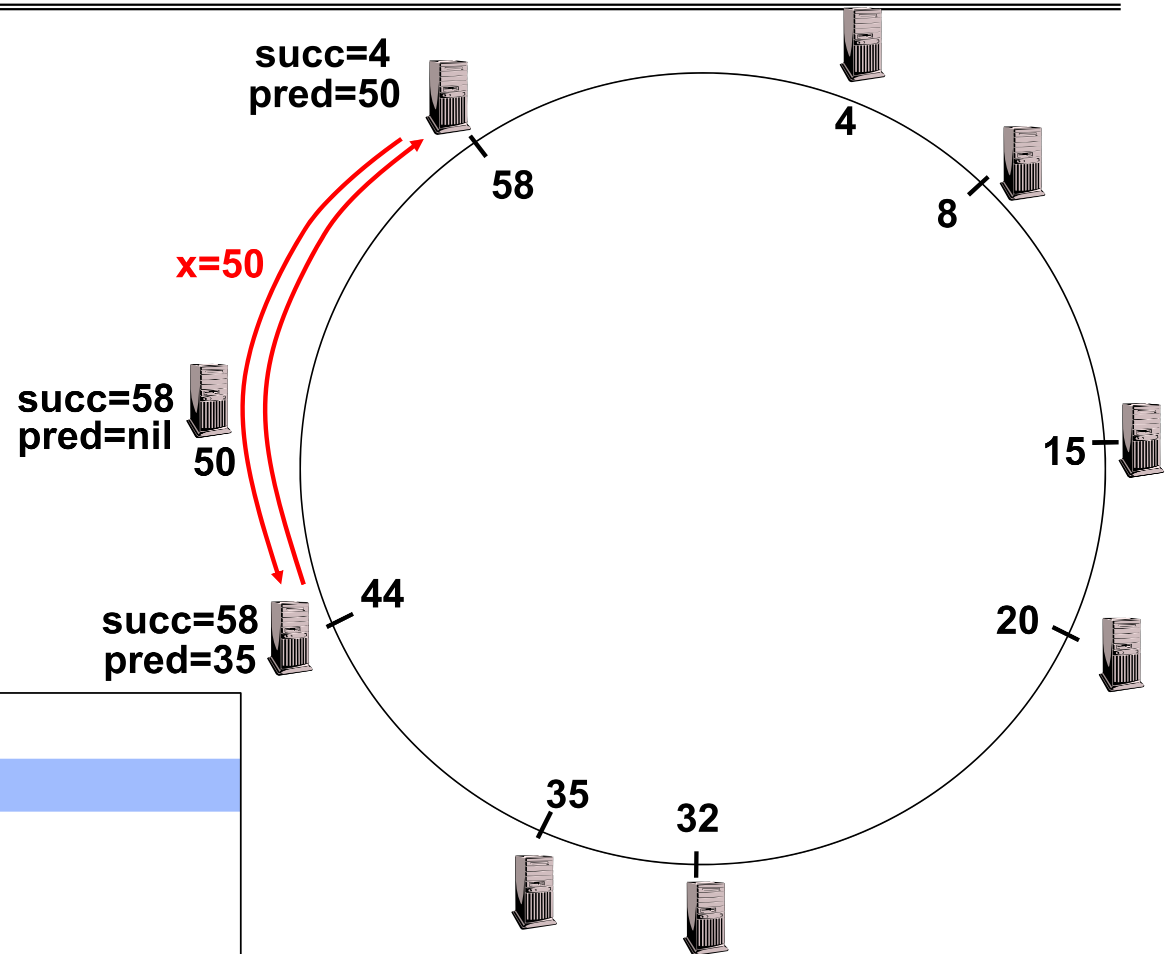
- n=58 executes notify(50)
  - pred = 44
  - n’ = 50
- set pred = 50




```
n.notify(n’)
  if (pred = nil or n’ ∈ (pred, n))
     pred = n’
```

Joining Operation

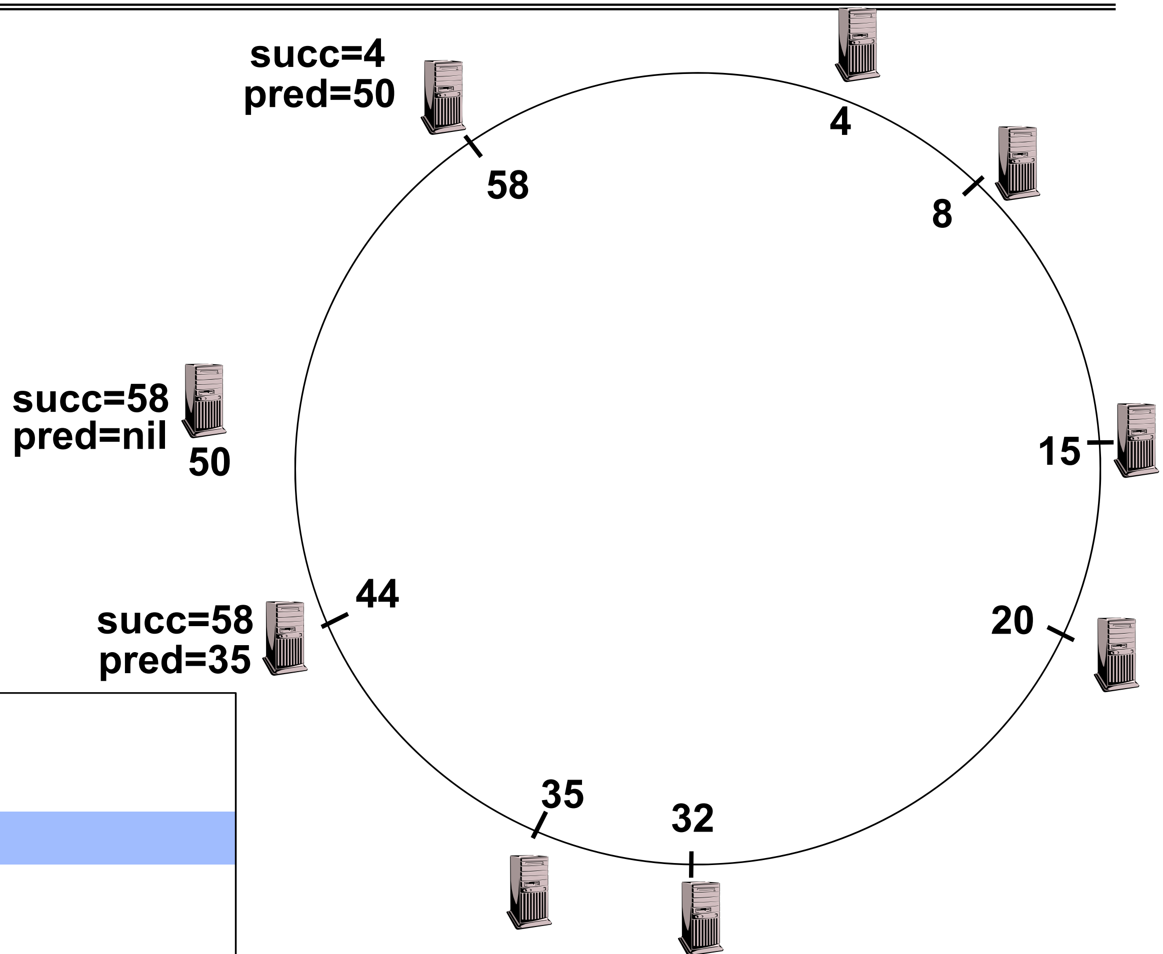
- n=44 executes stabilize()
- n's successor (58) returns x=50




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
     succ = x;
  succ.notify(n);
```

Joining Operation

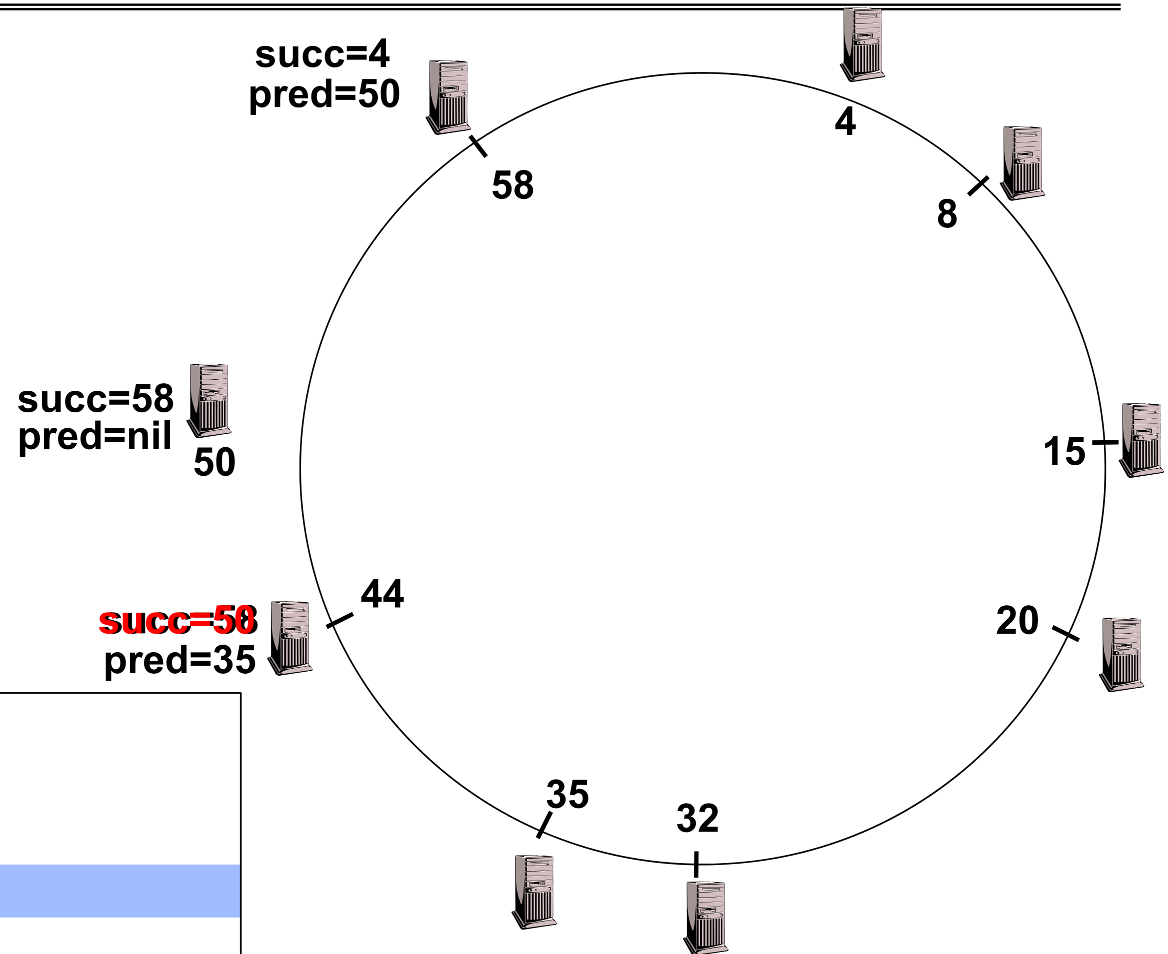
- n=44 executes stabilize()
  - x=50
  - succ=58




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
     succ = x;
  succ.notify(n);
```

Joining Operation

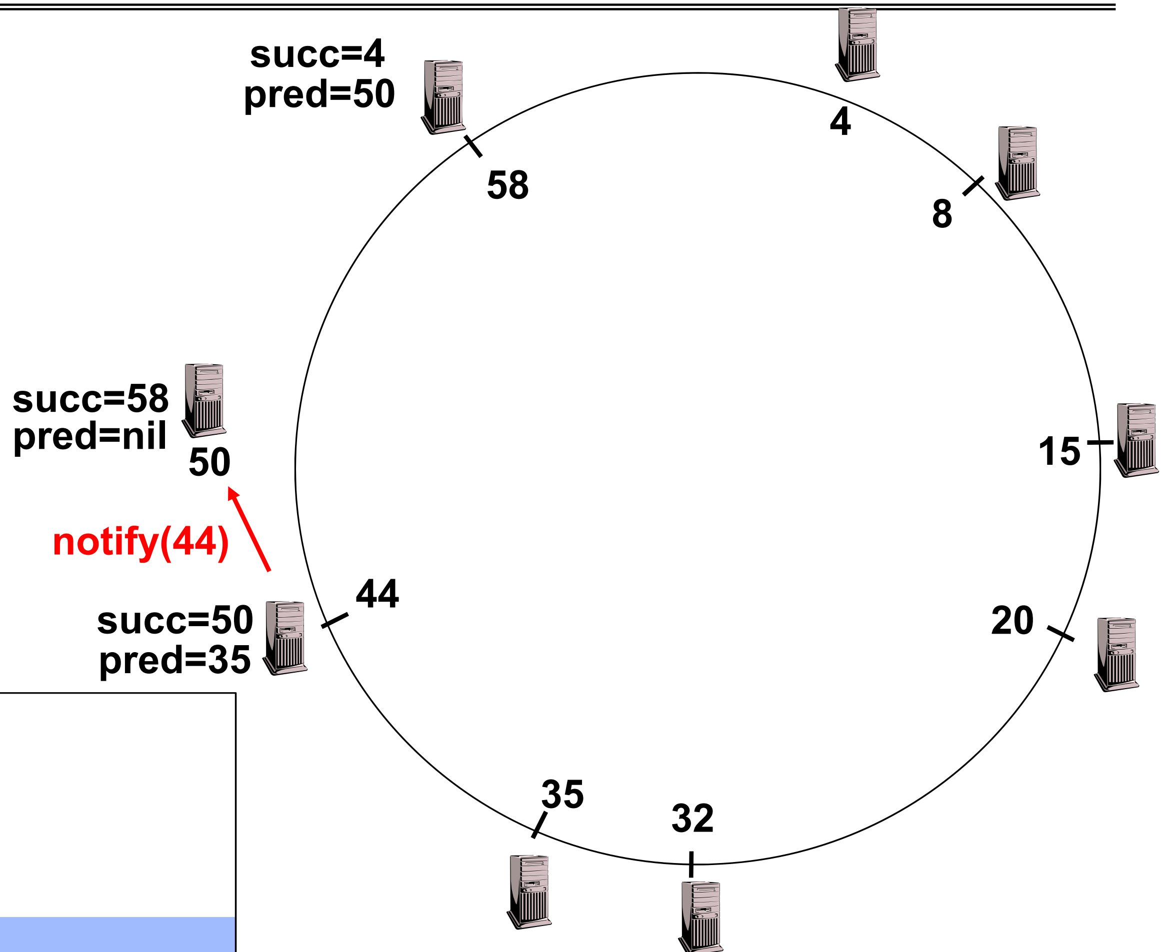
- n=44 executes stabilize()
  - x=50
  - succ=58
- n=44 sets succ=50




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
    succ = x;
  succ.notify(n);
```

Joining Operation

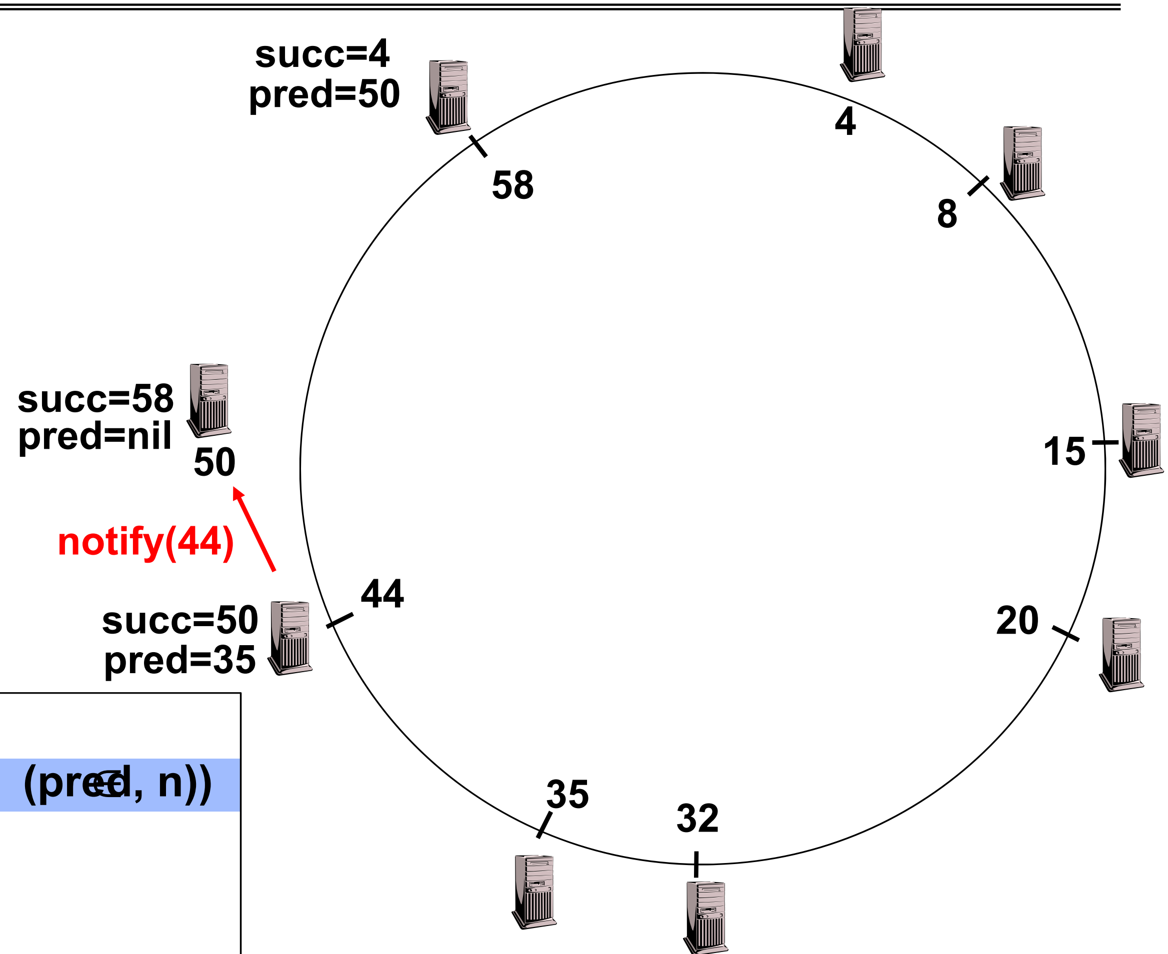
- n=44 executes stabilize()
- n=44 sends notify(44) to its successor




```
n.stabilize()
  x = succ.pred;
  if (x ∈(n, succ))
     succ = x;
  succ.notify(n);
```

Joining Operation

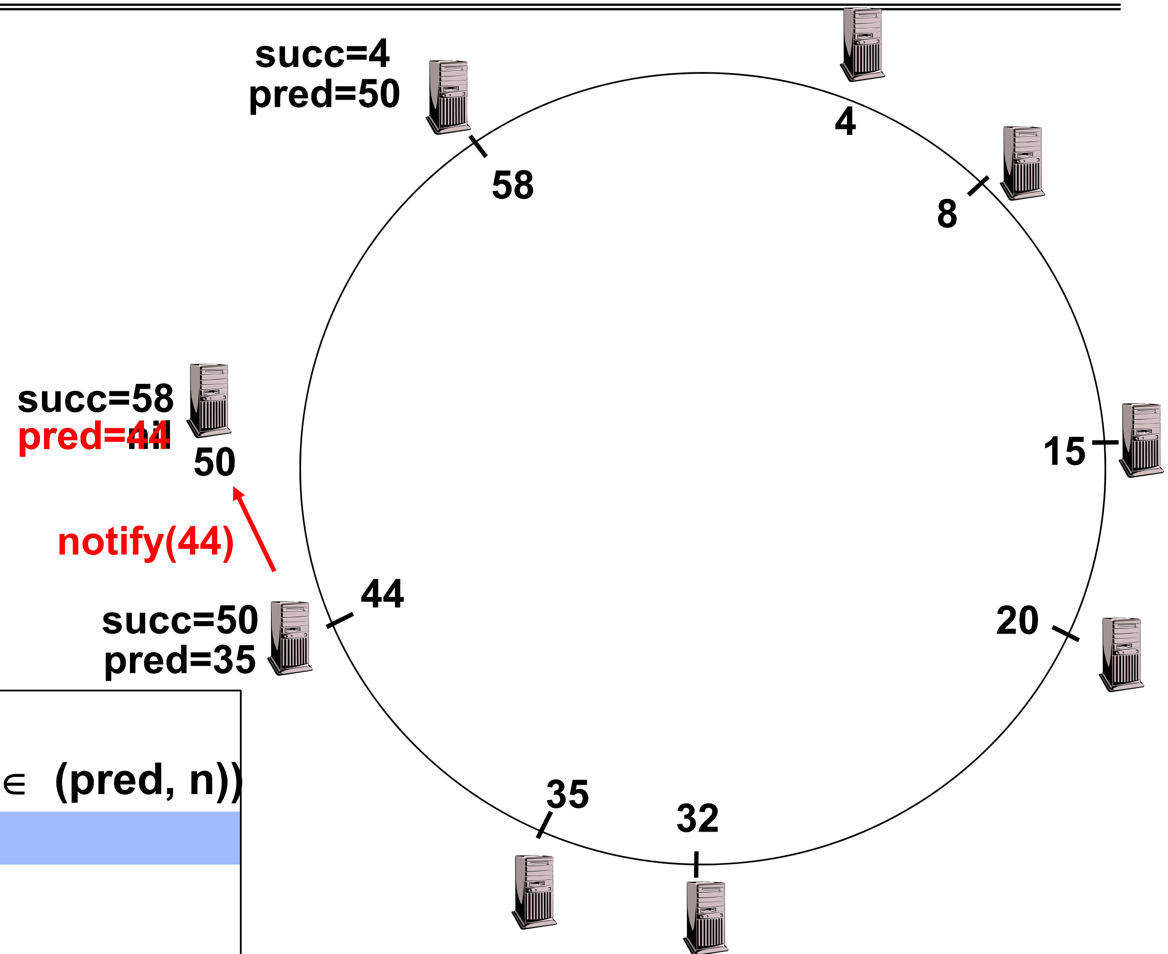
- n=50 executes notify(44)
  - pred=nil




```
n.notify(n')
  if (pred = nil or n' ∈ (pred, n))
    pred = n'
```

Joining Operation

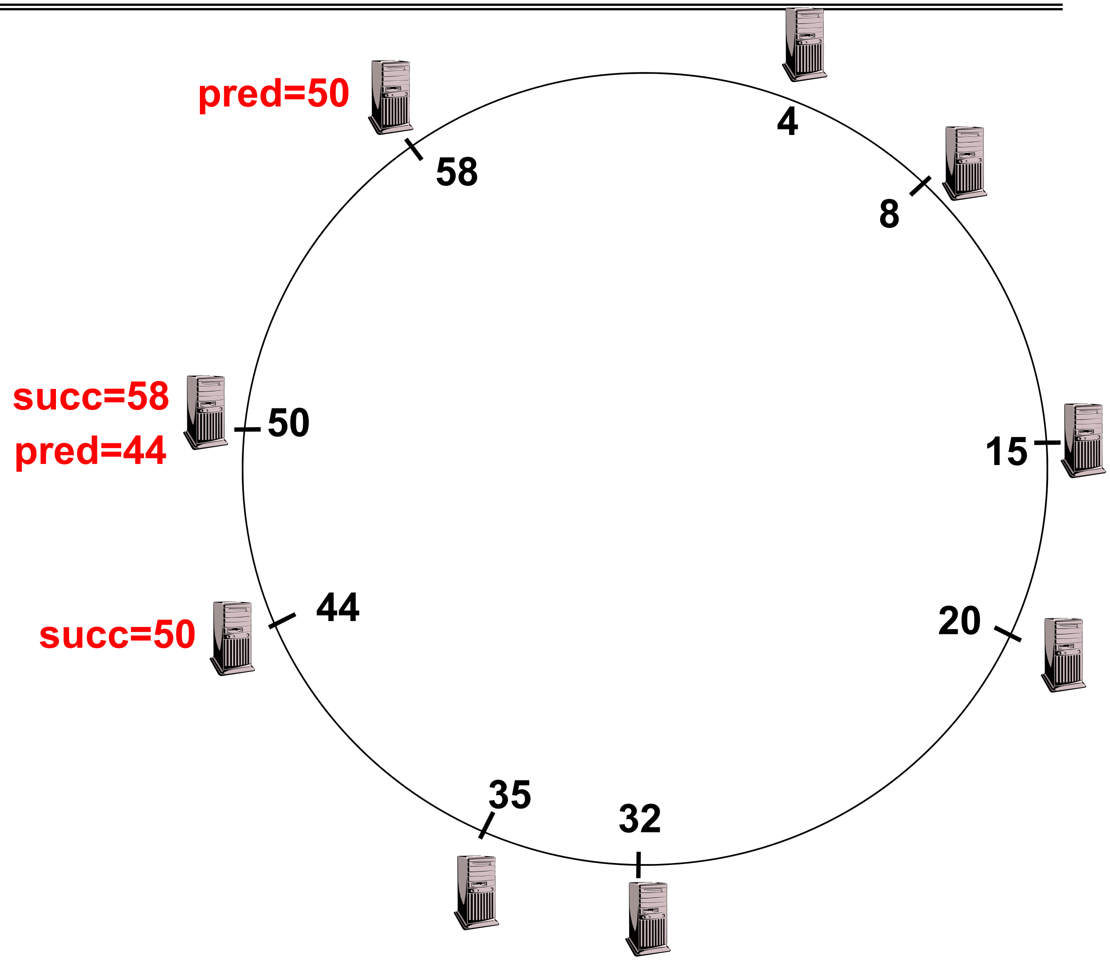
- n=50 executes notify(44)
  - pred=nil
- n=50 sets pred=44




```
n.notify(n')
  if (pred = nil or n' ∈ (pred, n))
     pred = n'
```

Joining Operation (cont'd)

- This completes the joining operation!
- The same stabilizing process will deal with failed nodes by reconnecting the ring
- What if 2 or more nodes in a row fail?
  - Keep track of more neighbors!
  - Called the "leaf set"

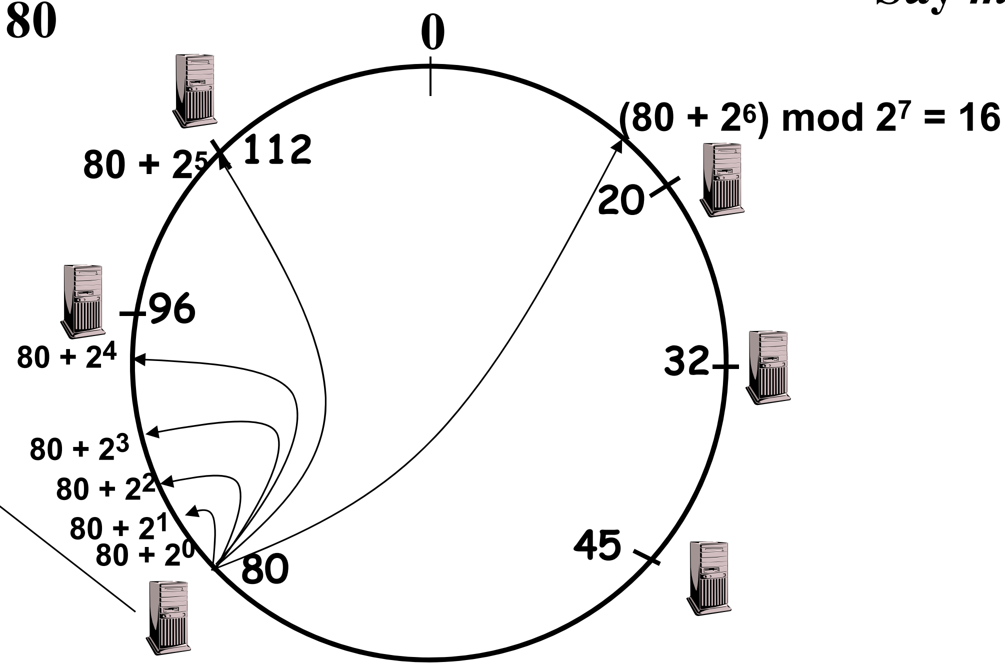


Achieving Efficiency: *finger tables*

Say *m=7*

Finger Table at 80

i	*ft[i]*
0	96
1	96
2	96
3	96
4	96
5	112
6	20



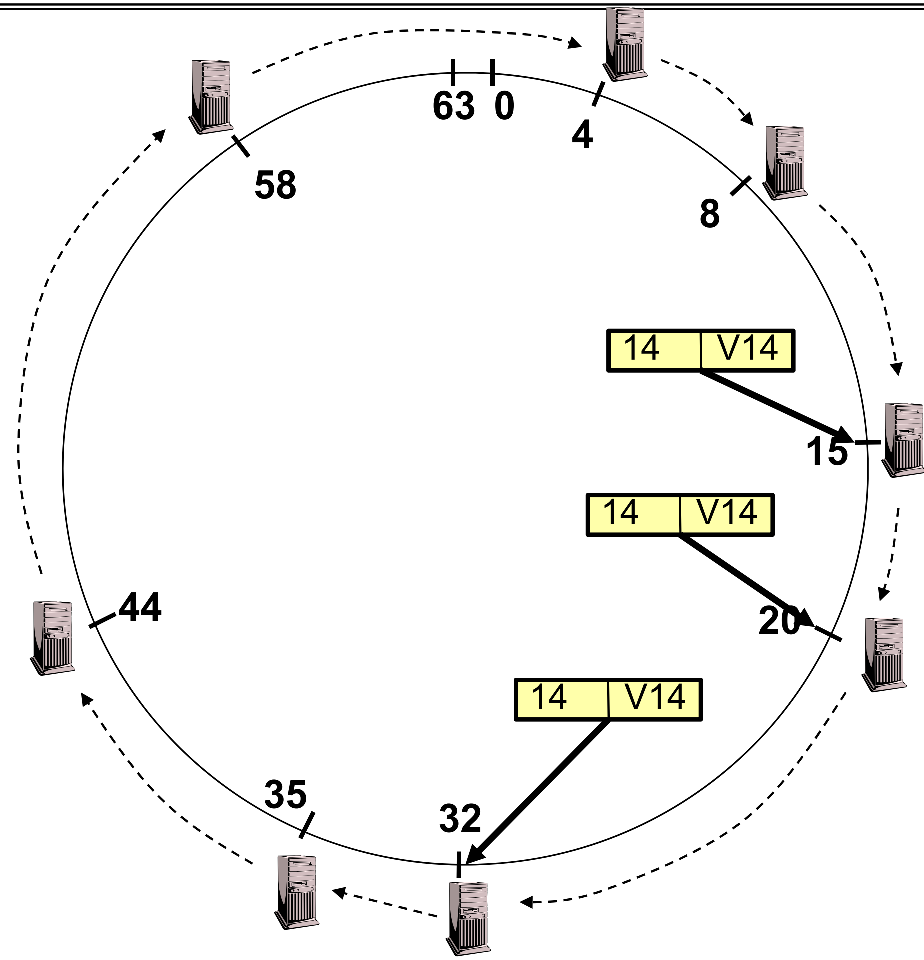
*i*th entry at peer with id *n* is first peer with id >= $n + 2^i \pmod{2^m}$

Achieving Fault Tolerance for Lookup Service

- To improve robustness each node maintains the k (> 1) immediate successors instead of only one successor
  - Again – called the “leaf set”
  - In the pred() reply message, node A can send its k-1 successors to its predecessor B
  - Upon receiving pred() message, B can update its successor list by concatenating the successor list received from A with its own list
- If k = log(M), lookup operation works with high probability even if half of nodes fail, where M is number of nodes in the system

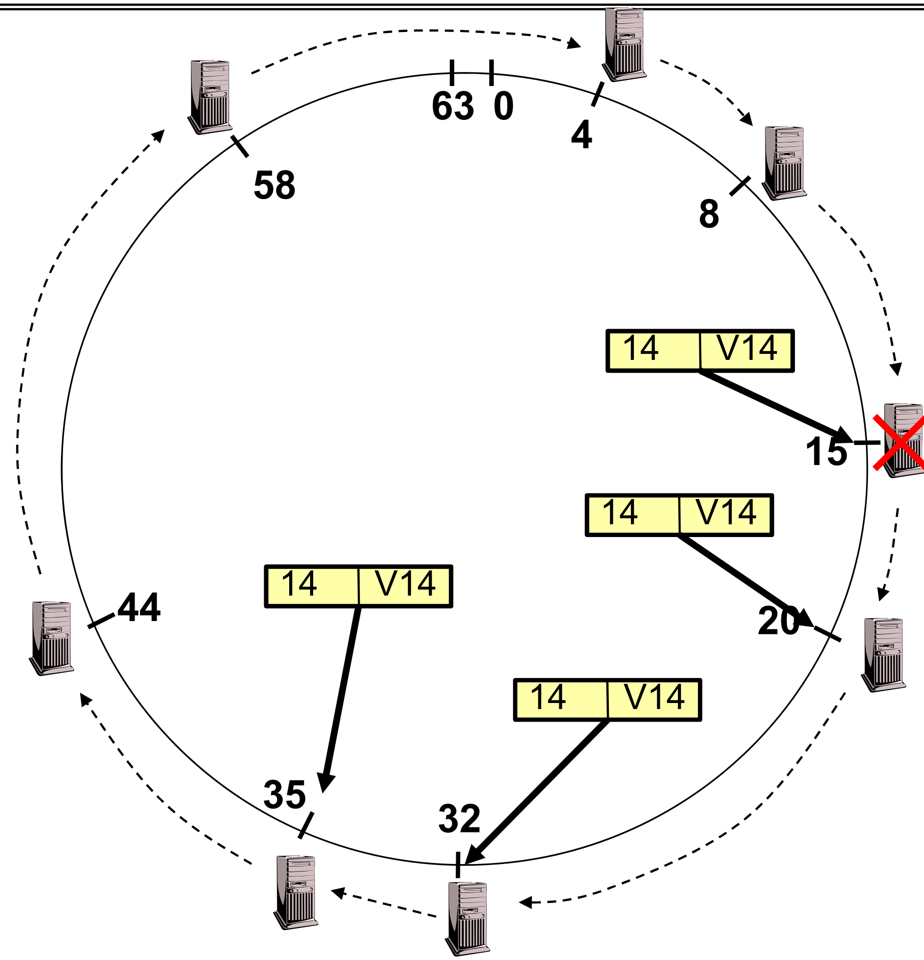
Storage Fault Tolerance

- Replicate tuples on successor nodes
- Example: replicate (K14, V14) on nodes 20 and 32

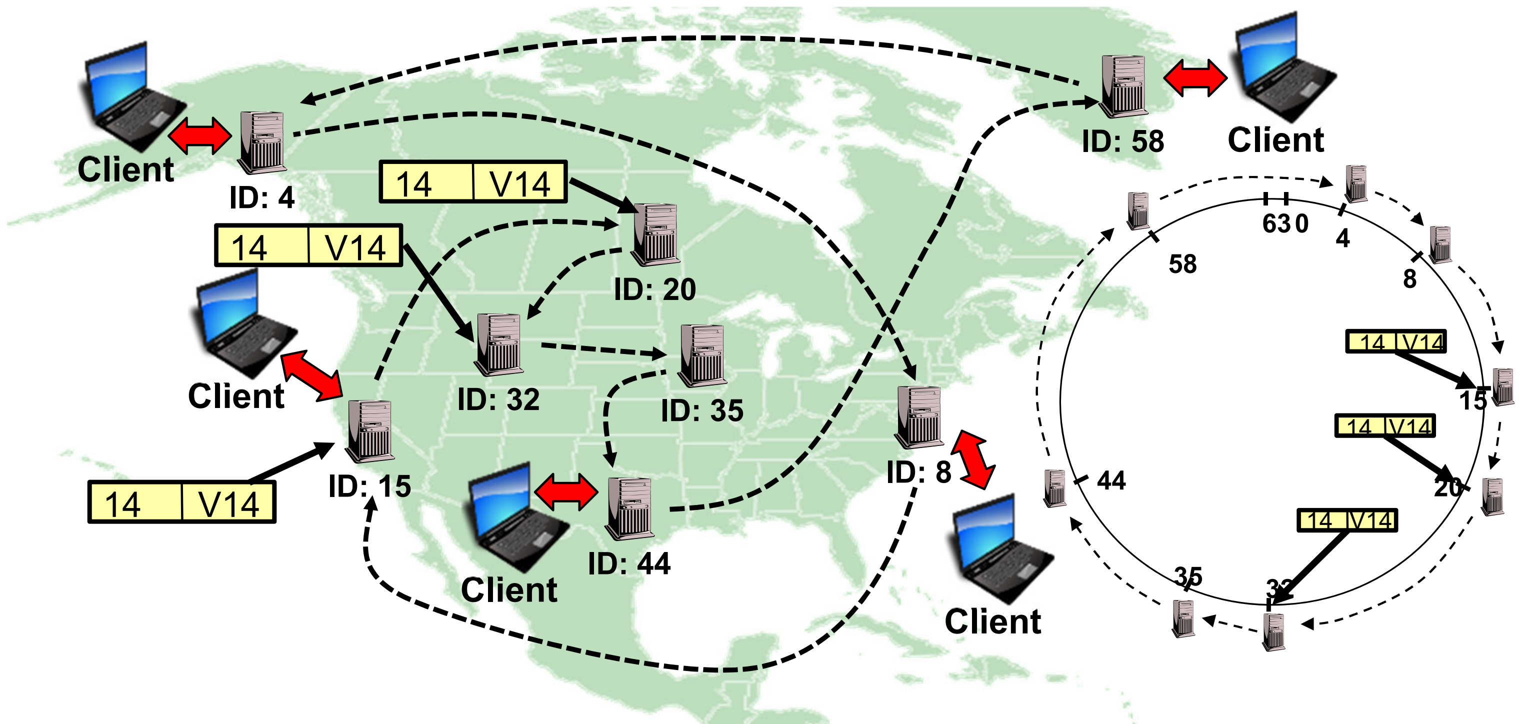


Storage Fault Tolerance

- If node 15 fails, no reconfiguration needed
  - Still have two replicas
  - All lookups will be correctly routed after stabilization
- Will need to add a new replica on node 35



Replication in Physical Space



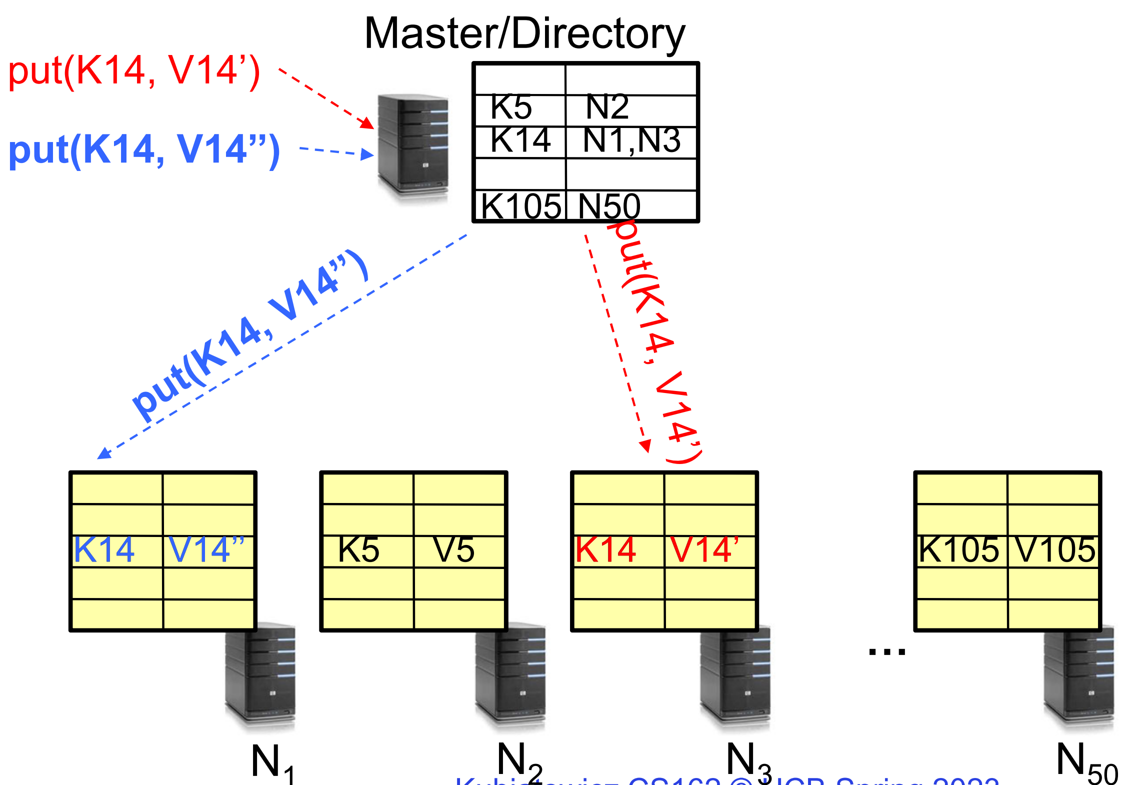
- Replicating in Adjacent nodes of virtual space ⇒ Geographic Separation in physical space
  - – Avoids single-points of failure through randomness
  - – More nodes, more replication, more geographic spread

Consistency

- Need to make sure that a value is replicated correctly
- How do you know a value has been replicated on every node?
  - Wait for acknowledgements from every node
- What happens if a node fails during replication?
  - Pick another node and try again
- What happens if a node is slow?
  - Slow down the entire put()? Pick another node?
- In general, with multiple replicas
  - Slow puts and fast gets

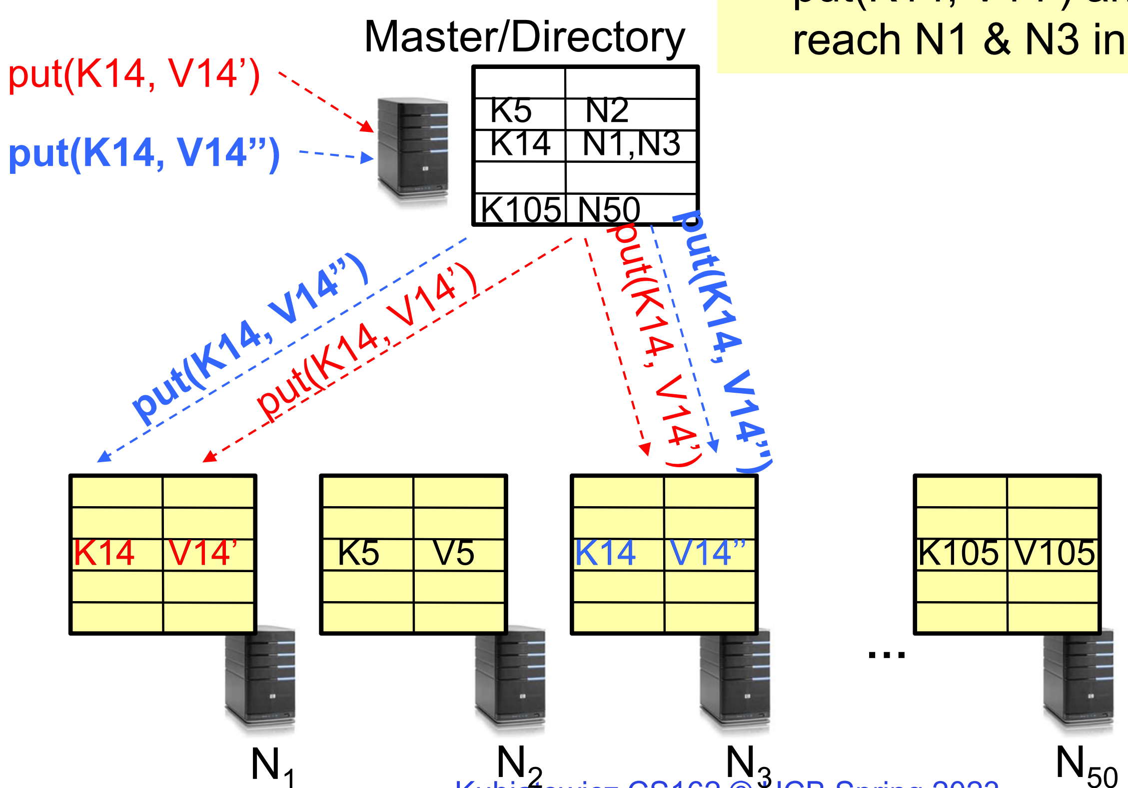
Consistency (cont'd)

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



Consistency (cont'd)

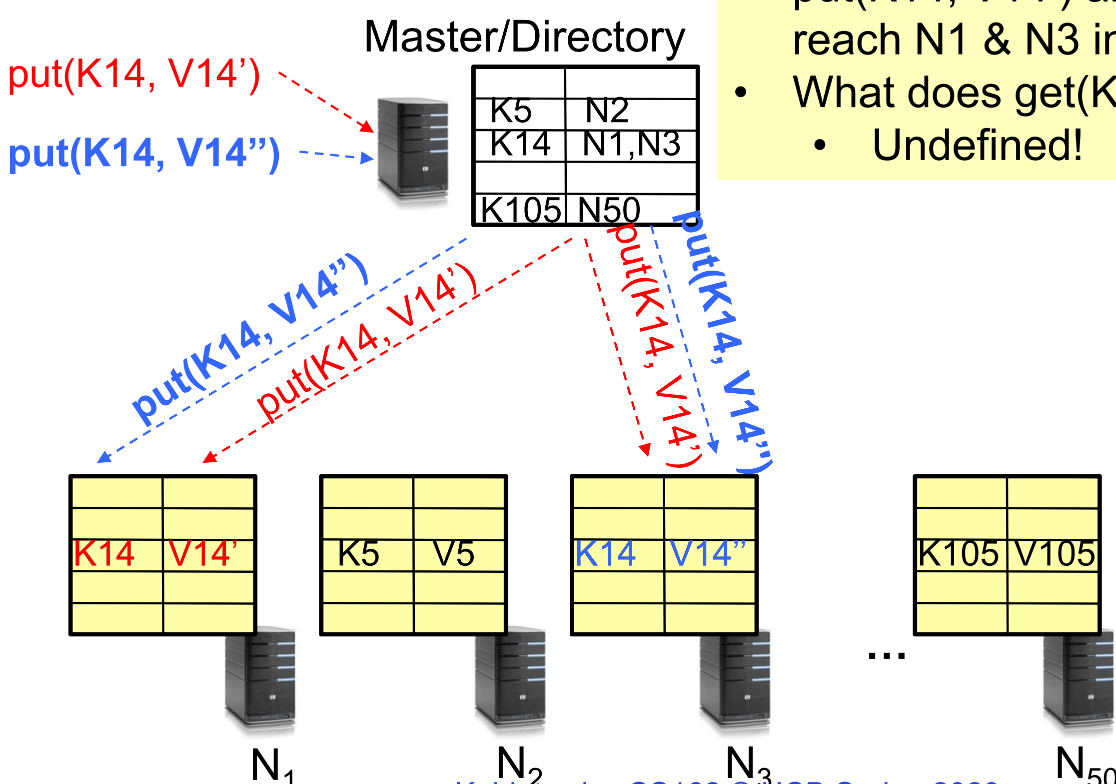
- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



- put(K14, V14') and put(K14, V14'') reach N1 & N3 in reverse order!

Consistency (cont'd)

- If concurrent updates (i.e., puts to same key) may need to make sure that updates happen in the same order



- put(K14, V14') and put(K14, V14'') reach N1 & N3 in reverse order!
- What does get(K14) return?
  - Undefined!

Large Variety of Consistency Models

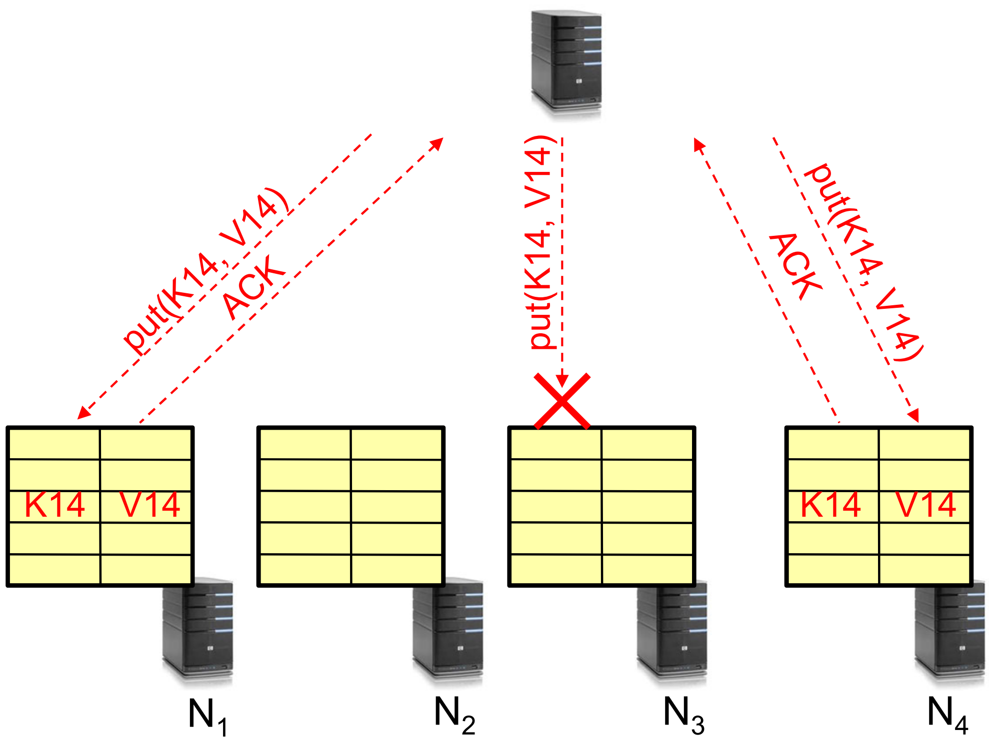
- Atomic consistency (linearizability): reads/writes (gets/puts) to replicas appear as if there was a single underlying replica (single system image)
  - Think “one updated at a time”
  - Transactions
- Eventual consistency: given enough time all updates will propagate through the system
  - One of the weakest form of consistency; used by many systems in practice
  - Must eventually converge on single value/key (coherence)
- And many others: causal consistency, sequential consistency, strong consistency, …

Quorum Consensus

- Improve put() and get() operation performance
  – In the presence of replication!
- Define a replica set of size N
  – put() waits for acknowledgements from at least W replicas
    » Different updates need to be differentiated by something monotonically increasing like a timestamp
    » Allows us to replace old values with updated ones
  – get() waits for responses from at least R replicas
  – W+R > N
- Why does it work?
  – There is at least one node that contains the update
- Why might you use W+R > N+1?

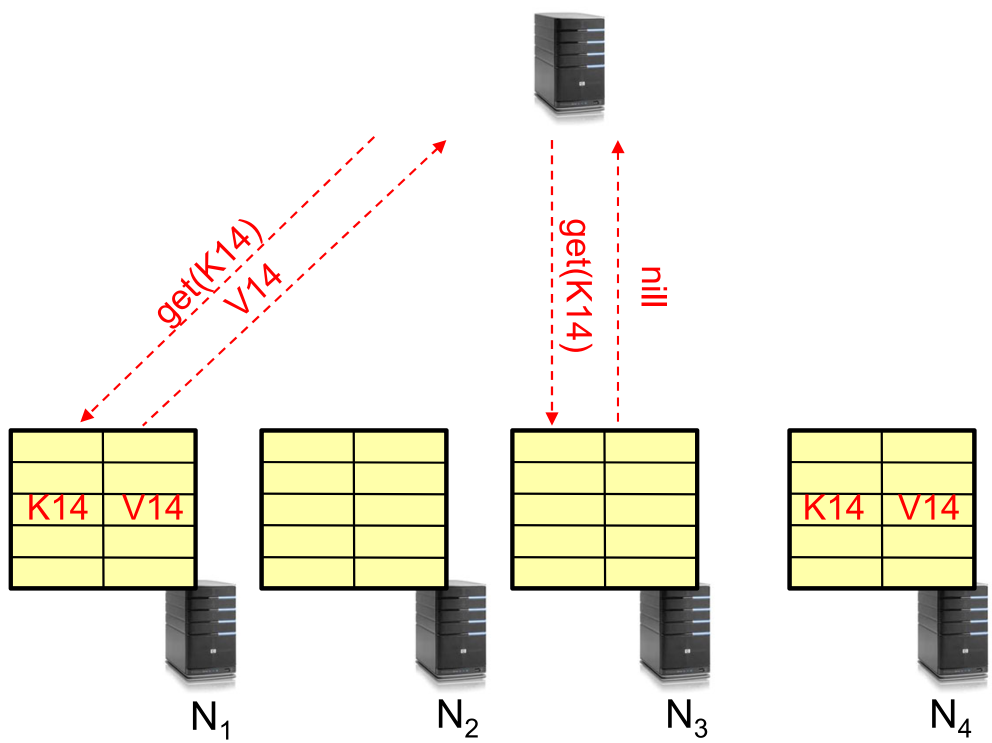
Quorum Consensus Example

- N=3, W=2, R=2
- Replica set for K14: {N1, N2, N4}
- Assume put() on N3 fails



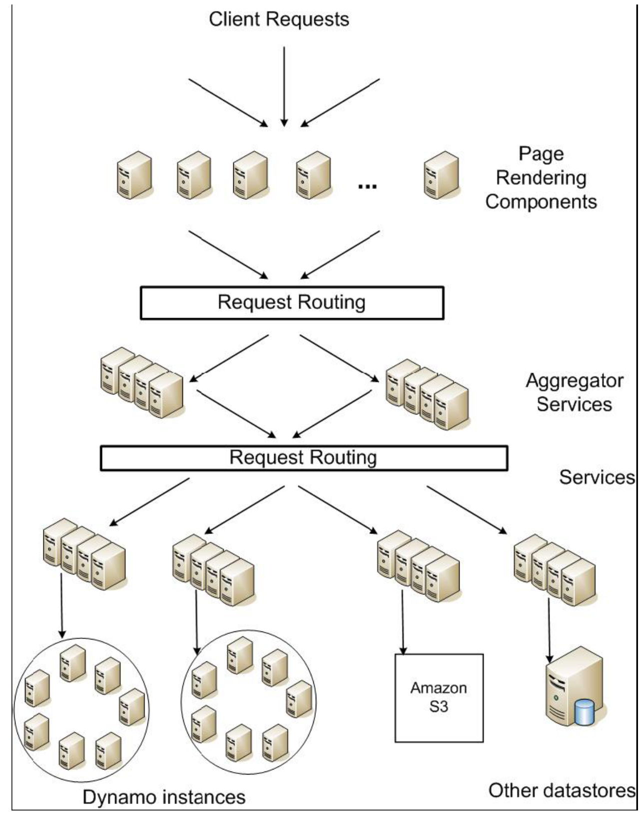
Quorum Consensus Example

- Now, issuing get() to any two nodes out of three will return the answer



DynamoDB Example: Service Level Agreements (SLA)

- Dynamo is Amazon's storage system using "Chord" ideas
- Application can deliver its functionality in a bounded time:
  - Every dependency in the platform needs to deliver its functionality with even tighter bounds.
- Example: service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second
- Contrast to services which focus on mean response time



Service-oriented architecture of Amazon's platform

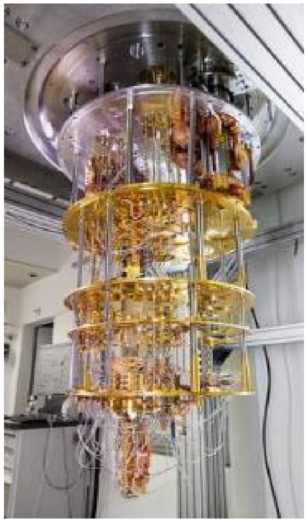
Quantum Computing, Shor's Algorithm, and the role of CAD design

Use Quantum Mechanics to Compute?

- Weird but useful properties of quantum mechanics:
  - Quantization: Only certain values or orbits are good
    - » Remember orbitals from chemistry???
  - Superposition: Schizophrenic physical elements don't quite know whether they are one thing or another
- All existing digital abstractions try to eliminate QM
  - Transistors/Gates designed with classical behavior
  - Binary abstraction: a "1" is a "1" and a "0" is a "0"
- Quantum Computing:
  Use of Quantization and Superposition to compute.
- Interesting results:
  - Shor's algorithm: factors in polynomial time!
  - Grover's algorithm: Finds items in unsorted database in time proportional to square-root of n.
  - Materials simulation: exponential classically, linear-time QM

Current “Arms Race” of Quantum Computing

Google: Superconducting
Devices up 72-qubits

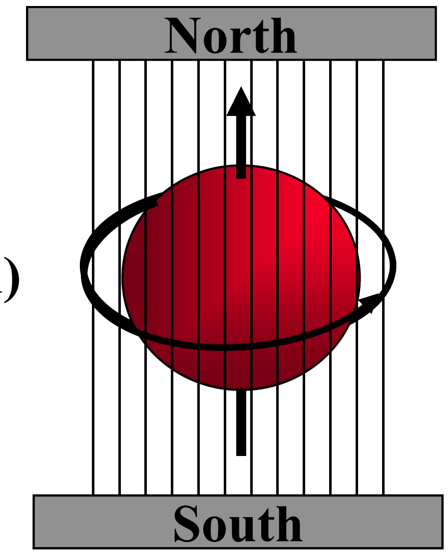


IBM: Superconducting
Devices up to 50 qubits

- Big companies looking at Quantum Computing Seriously
  - Google, IBM, Microsoft
- Current Goal: Quantum Supremacy
  - Show that Quantum Computers faster than Classical ones
  - “If a quantum processor can be operated with low enough error, it would be able to outperform a classical supercomputer on a well-defined computer science problem, an achievement known as quantum supremacy.”

Quantization: Use of “Spin”

Spin ½ particle:
(Proton/Electron)



Representation:
|0> or |1>

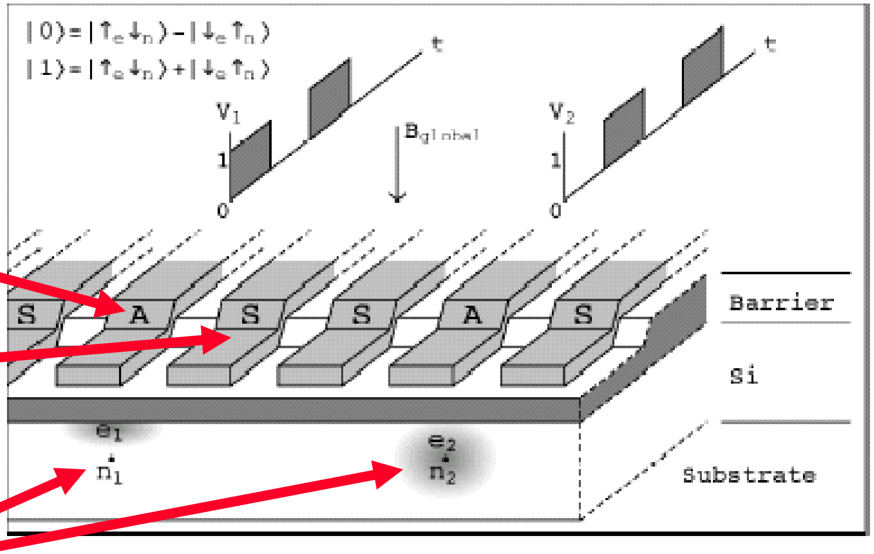
- Particles like Protons have an intrinsic “Spin” when defined with respect to an external magnetic field
- Quantum effect gives “1” and “0”:
  - Either spin is “UP” or “DOWN” nothing between

Kane Proposal II
(First one didn't quite work)

Single Spin Control Gates

Inter-bit Control Gates

Phosphorus Impurity Atoms



- Bits Represented by combination of proton/electron spin
- Operations performed by manipulating control gates
  - Complex sequences of pulses perform NMR-like operations
- Temperature < 1° Kelvin!

Now add Superposition!

- The bit can be in a combination of "1" and "0":
  - Written as: $\Psi = C_0|0\rangle + C_1|1\rangle$
  - The C's are *complex numbers!*
  - Important Constraint: $|C_0|^2 + |C_1|^2 = 1$
- If *measure* bit to see what looks like,
  - With probability $|C_0|^2$ we will find $|0\rangle$ (say "UP")
  - With probability $|C_1|^2$ we will find $|1\rangle$ (say "DOWN")
- Is this a real effect? Options:
  - This is just statistical – given a large number of protons, a fraction of them ($|C_0|^2$) are "UP" and the rest are down.
  - This is a real effect, and the proton is really both things until you try to look at it
- Reality: second choice!
  - There are experiments to prove it!

A register can have many values!

- Implications of superposition:
  - An *n*-bit register can have 2^n values simultaneously!
  - 3-bit example:

    $\Psi = C_{000}|000\rangle + C_{001}|001\rangle + C_{010}|010\rangle + C_{011}|011\rangle +$
    $C_{100}|100\rangle + C_{101}|101\rangle + C_{110}|110\rangle + C_{111}|111\rangle$
- Probabilities of measuring all bits are set by coefficients:
  - So, prob of getting $|000\rangle$ is $|C_{000}|^2$, etc.
  - Suppose we measure only one bit (first):
    - » We get "0" with probability: $P_0=|C_{000}|^2+|C_{001}|^2+|C_{010}|^2+|C_{011}|^2$
      Result: $\Psi = (C_{000}|000\rangle + C_{001}|001\rangle + C_{010}|010\rangle + C_{011}|011\rangle)$
    - » We get "1" with probability: $P_1=|C_{100}|^2+|C_{101}|^2+|C_{110}|^2+|C_{111}|^2$
      Result: $\Psi = (C_{100}|100\rangle + C_{101}|101\rangle + C_{110}|110\rangle + C_{111}|111\rangle)$
- Problem: Don't want environment to *measure* before ready!
  - Solution: Quantum Error Correction Codes!

Spooky action at a distance

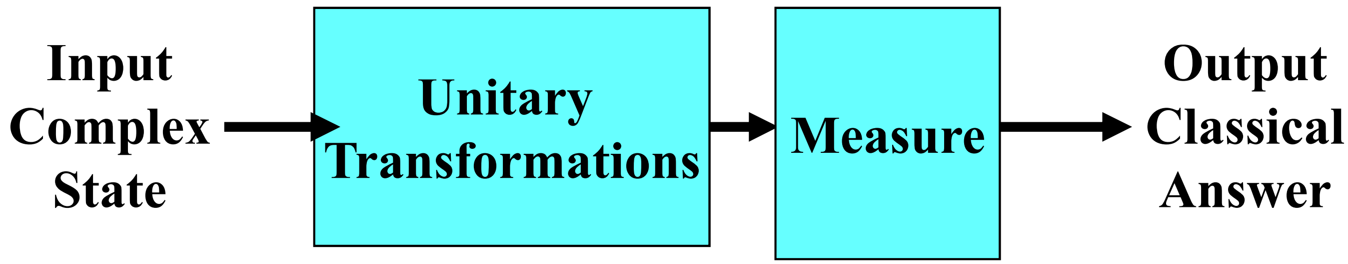
- Consider the following simple 2-bit state:

  $\Psi = C_{00}|00\rangle + C_{11}|11\rangle$

  – Called an "EPR" pair for "Einstein, Podolsky, Rosen"
- Now, separate the two bits:

- If we measure one of them, it instantaneously sets other one!
  – Einstein called this a "spooky action at a distance"
  – In particular, if we measure a |0> at one side, we get a |0> at the other (and vice versa)
- Teleportation
  – Can "pre-transport" an EPR pair (say bits X and Y)
  – Later to transport bit A from one side to the other we:
    » Perform operation between A and X, yielding two classical bits
    » Send the two bits to the other side
    » Use the two bits to operate on Y
    » Poof! State of bit A appears in place of Y

Model: Operations on coefficients + measurements



- Basic Computing Paradigm:
  - Input is a register with superposition of many values
    - » Possibly all 2n inputs equally probable!
  - Unitary transformations compute on coefficients
    - » Must maintain probability property (sum of squares = 1)
    - » Looks like doing computation on all 2n inputs simultaneously!
  - Output is one result attained by measurement
- If do this poorly, just like probabilistic computation:
  - If 2n inputs equally probable, may be 2n outputs equally probable.
  - After measure, like picked random input to classical function!
  - All interesting results have some form of "fourier transform" computation being done in unitary transformation

Shor's Factoring Algorithm

- The Security of RSA Public-key cryptosystems depends on the difficulty of factoring a number N=pq (product of two primes)
  - Classical computer: sub-exponential time factoring
  - Quantum computer: polynomial time factoring
- Shor's Factoring Algorithm (for a quantum computer)

Easy 1) Choose random x : $2 \leq x \leq N$-1.

Easy 2) If gcd(x,N) $\neq$ 1, Bingo!

Hard 3) Find smallest integer r : $x^r \equiv 1 \pmod{N}$

Easy 4) If r is odd, Repeat at Step 1

Easy 5) If r is even, $a \equiv x^{r/2} \pmod{N} \Rightarrow (a\text{-}1)\times(a+1) = kN$

Easy 6) If $a \equiv N$-1(mod N) GOTO 1

Easy 7) ELSE gcd($a \pm 1,N$) is a non trivial factor of N.

Finding r with $x^r \equiv 1 \pmod{N}$

$$\sum_k |k\rangle\,|1\rangle \;\longrightarrow\; \sum_k |k\rangle\,|\cancel{x}^k\rangle$$

$$= \sum_{w=0}^{r-1} \sum_{y} |w + r\,y\rangle\,|\cancel{x}^w\rangle$$

Quantum Fourier Transform

$$\longmapsto \sum_{w=0}^{r-1} \Big(\quad \Big)\,|\cancel{x}^w\rangle$$

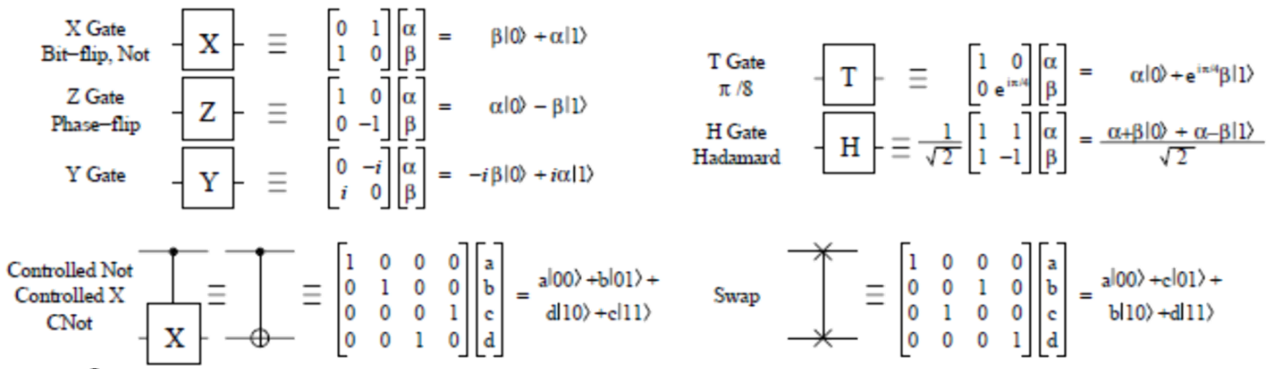
$\frac{0}{r}$ $\frac{1}{r}$ $\frac{k}{r}$

- Finally: Perform measurement
  - Find out r with high probability
  - Get |y>|a$^{w'}$> where y is of form k/r and w' is related

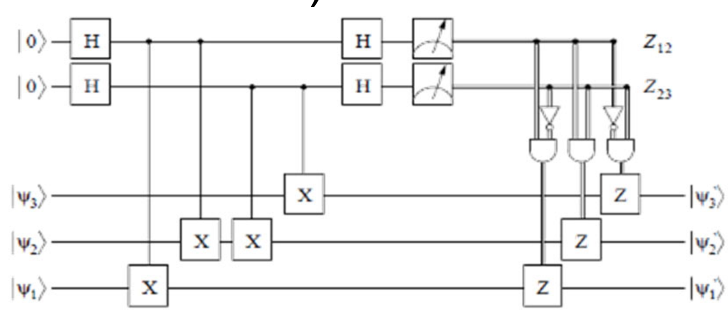
Quantum Computing Architectures

- Why study quantum computing?
  - Interesting, says something about physics
    - » Failure to build ⇒ quantum mechanics wrong?
  - Mathematical Exercise (perfectly good reason)
  - Hope that it will be practical someday:
    - » Shor's factoring, Grover's search, Design of Materials
    - » Quantum Co-processor included in your Laptop?
- To be practical, will need to hand quantum computer design off to classical designers
  - Baring Adiabatic algorithms, will probably need 100s to 1000s (millions?) of working logical Qubits ⇒
    1000s to millions of physical Qubits working together
  - Current chips: ~1 billion transistors!
- Large number of components is realm of *architecture*
  - What are optimized structures of quantum algorithms when they are mapped to a physical substrate?
  - Optimization not possible by hand
    - » Abstraction of elements to design larger circuits
    - » Lessons of last 30 years of VLSI design: USE CAD

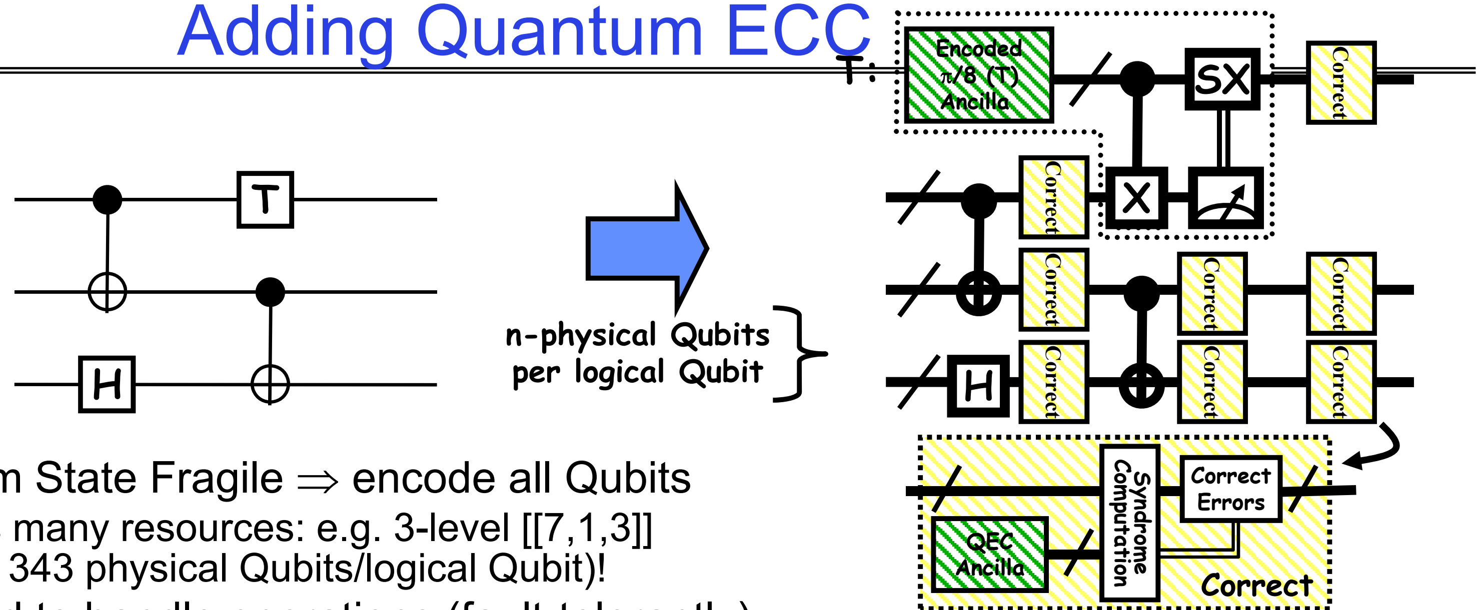
Quantum Circuit Model



- Quantum Circuit model – graphical representation
  – Time Flows from left to right
  – Single Wires: persistent Qubits, Double Wires: classical bits
    » Qubit – coherent combination of 0 and 1: $\psi = \alpha|0\rangle + \beta|1\rangle$
  – Universal gate set: Sufficient to form all unitary transformations
- Example: Syndrome Measurement (for 3-bit code)
  – Measurement (meter symbol) produces classical bits
- Quantum CAD
  – Circuit expressed as netlist
  – Computer manpulated circuits and implementations



Adding Quantum ECC



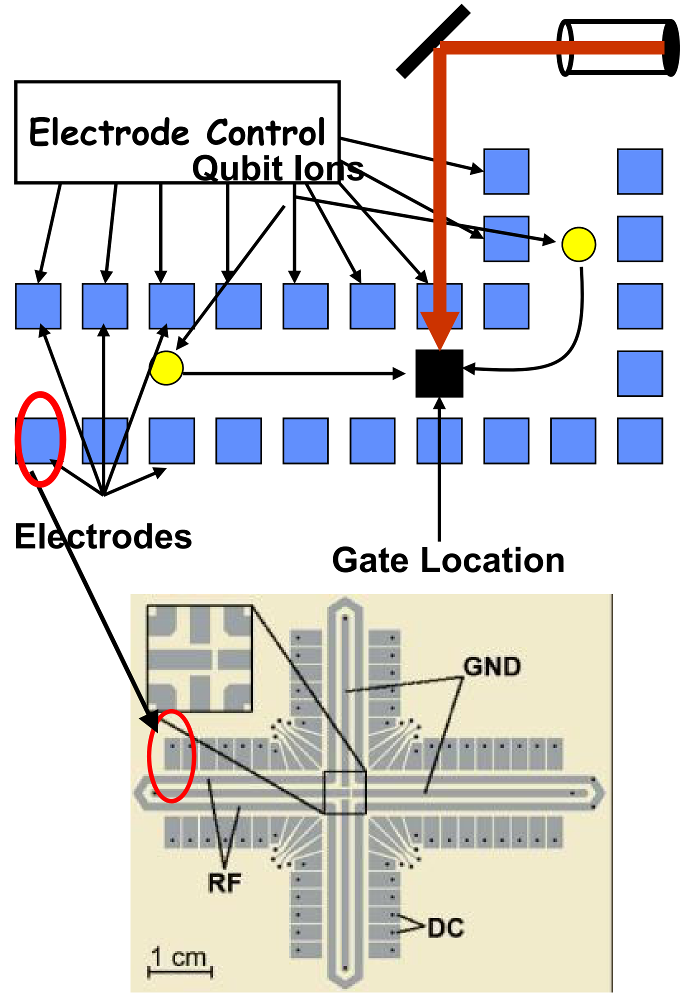
- Quantum State Fragile ⇒ encode all Qubits
  – Uses many resources: e.g. 3-level [[7,1,3]] code 343 physical Qubits/logical Qubit)!
- Still need to handle operations (fault-tolerantly)
  – *Some* set of gates are simply "transversal:"
    » Perform identical gate between each physical bit of logical encoding
  – Others (like T gate for [[7,1,3]] code) cannot be handled transversally
    » Can be performed fault-tolerantly by preparing appropriate ancilla
- Finally, need to perform periodical error correction
  – Correct after every(?): Gate, Long distance movement, Long Idle Period
  – Correction reducing entropy ⇒ Consumes Ancilla bits
- Observation: ≥ 90% of QEC gates are used for ancilla production
  ≥ 70-85% of all gates are used for ancilla production

MEMs-Based Ion Trap Devices

- Ion Traps: One of the more promising quantum computer implementation technologies
  - Built on Silicon
    - » Can bootstrap the vast infrastructure that currently exists in the microchip industry
  - Seems to be on a "Moore's Law" like scaling curve
    - » Many researchers working on this problem
  - Some optimistic researchers speculate about room temperature
- Properties:
  - Has a long-distance Wire
    - » So-called "ballistic movement"
  - Seems to have relatively long decoherence times
  - Seems to have relatively low error rates for:
    - » Memory, Gates, Movement

Quantum Computing with Ion Traps

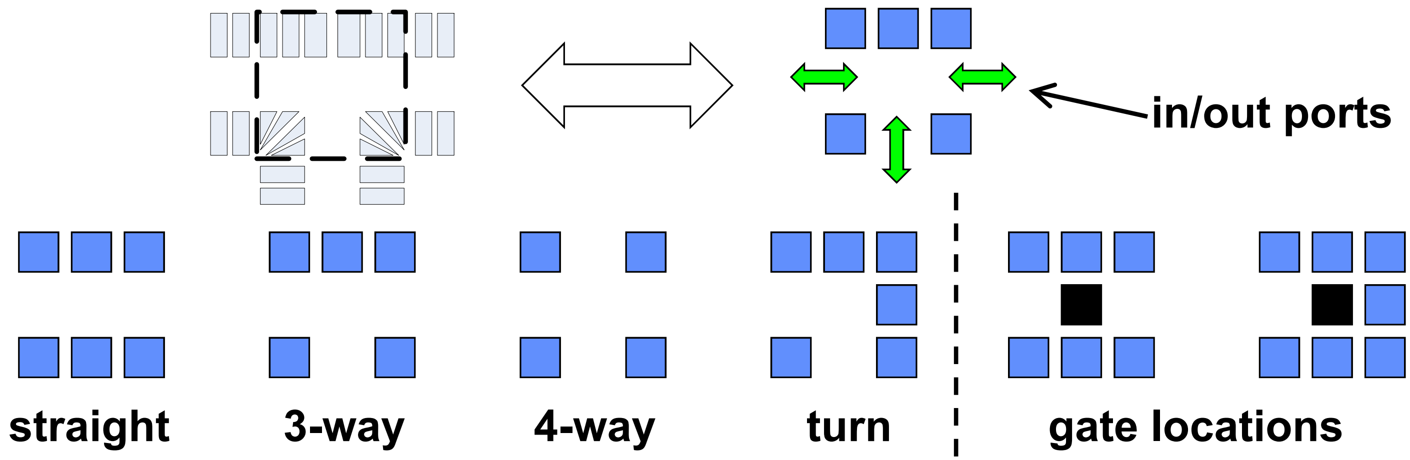
- Qubits are atomic ions (e.g. Be^+)
  - State is stored in hyperfine levels
  - Ions suspended in channels between electrodes
- Quantum gates performed by lasers (either one or two bit ops)
  - Only at certain trap locations
  - Ions move between laser sites to perform gates
- Classical control
  - Gate (laser) ops
  - Movement (electrode) ops
    - Complex pulse sequences to cause Ions to migrate
    - Care must be taken to avoid disturbing state
- Demonstrations in the Lab
  - NIST, MIT, Michigan, many others



Courtesy of Chuang group, MIT

An Abstraction of Ion Traps

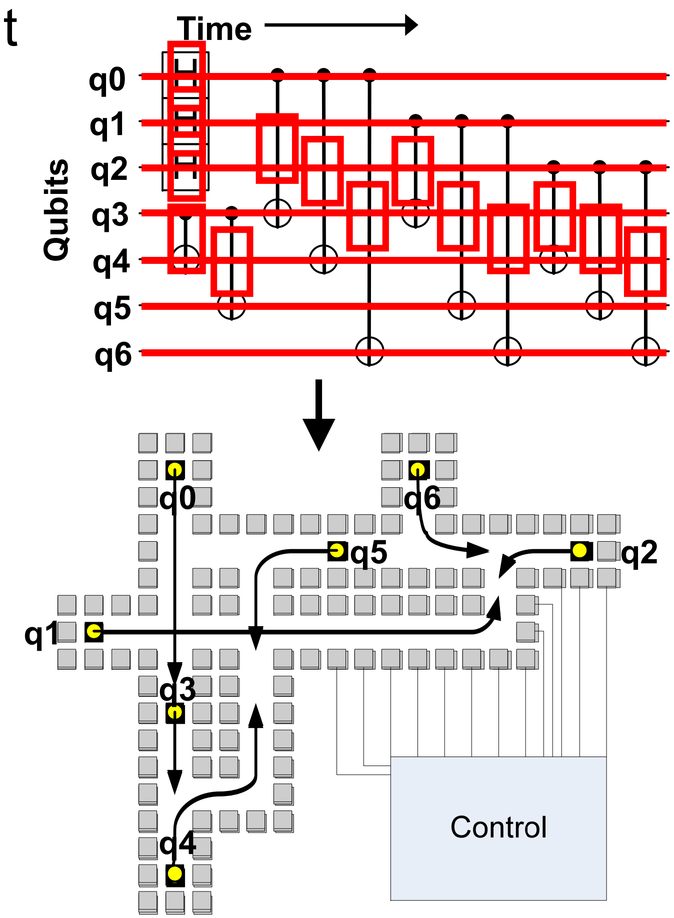
- *Basic block* abstraction: Simplify Layout



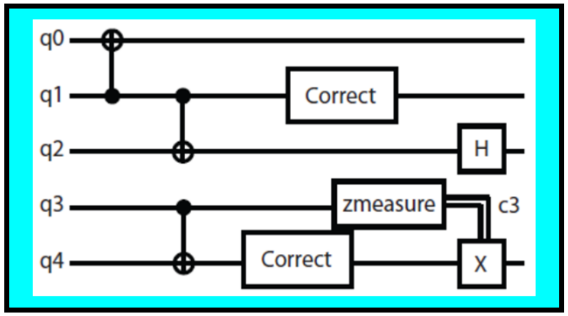
- Evaluation of layout through simulation
  - Yields Computation Time and Probability of Success
- Simple Error Model: Depolarizing Errors
  - Errors for every Gate Operation and Unit of Waiting
  - Ballistic Movement Error: Two error Models
    1. Every Hop/Turn has probability of error
    2. Only Accelerations cause error

Ion Trap Physical Layout

- Input: Gate level quantum circuit
  - Bit lines
  - 1-qubit gates
  - 2-qubit gates
- Output:
  - Layout of channels
  - Gate locations
  - Initial locations of ions
  - Movement/gate schedule
  - Control for schedule



Vision of Quantum Circuit Design



Schematic Capture
(Graphical Entry)

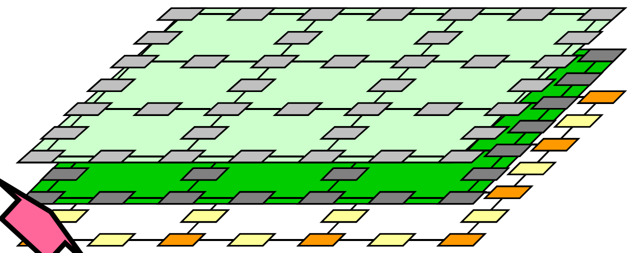
OR

```
cx q1, q0;
cx q1, q2;
correct q1;
h q2;
cx q3, q4;
zmeasure q3, c3;
correct q4;
(@c3==1) x q4;
```

Quantum Assembly
(QASM)

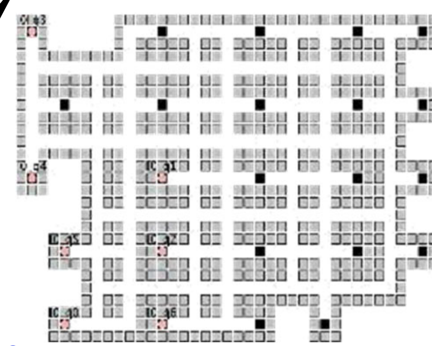
QEC Insertion
Partitioning
Layout
Network Insertion
Error Analysis
…
Optimization

CAD Tool
Implementation



Classical Control
Teleportation Network

Custom Layout and
Scheduling



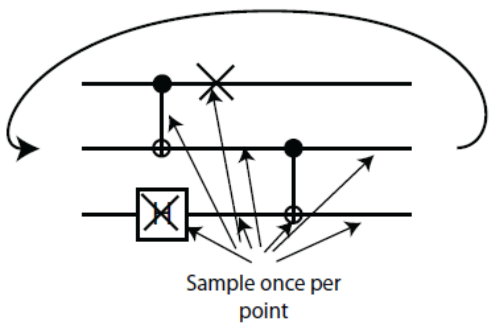
Important Measurement Metrics

- Traditional CAD Metrics:
  - Area
    - » What is the total area of a circuit?
    - » Measured in macroblocks (ultimately μm^2 or similar)
  - Latency ($Latency_{single}$)
    - » What is the total latency to compute circuit *once*
    - » Measured in seconds (or μs)
  - Probability of Success ($P_{success}$)
    - » Not common metric for classical circuits
    - » Account for occurrence of errors and error correction
- Quantum Circuit Metric: ADCR
  - Area-Delay to Correct Result: Probabilistic Area-Delay metric
  - ADCR = Area × E(Latency) = $\dfrac{Area \times Latency_{single}}{P_{success}}$
  - $ADCR_{optimal}$: Best ADCR over all configurations
- Optimization potential: Equipotential designs
  - Trade Area for lower latency
  - Trade lower probability of success for lower latency

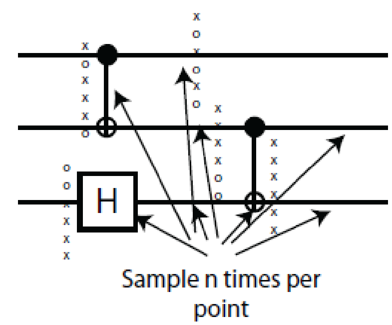
How to evaluate a circuit?

- First, generate a physical instance of circuit
  - Encode the circuit in one or more QEC codes
  - Partition and layout circuit: Highly dependant of layout heuristics!
    - » Create a physical layout and scheduling of bits
    - » Yields area and communication cost

Normal Monte Carlo: n times

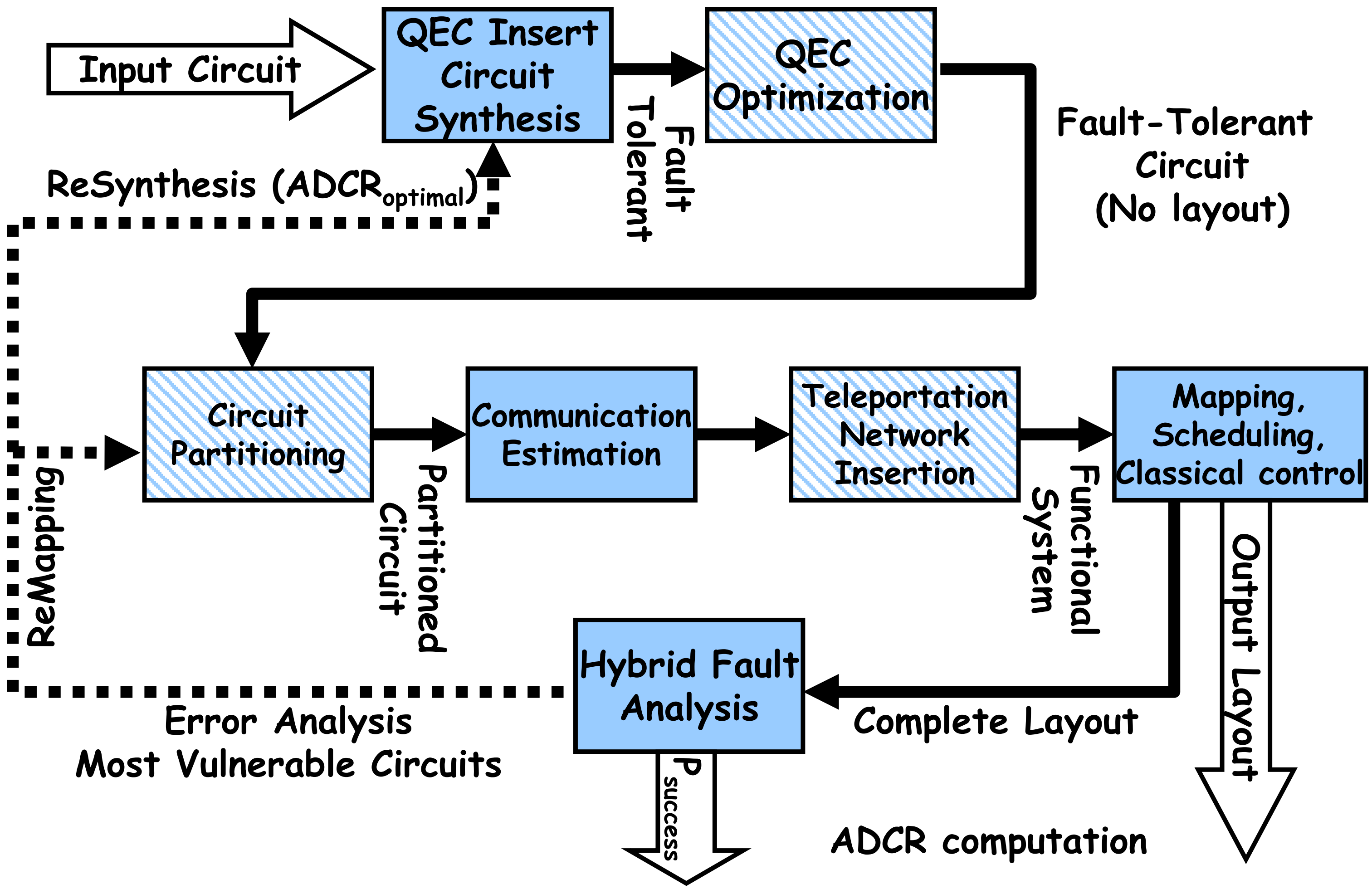


Vector Monte Carlo: single pass

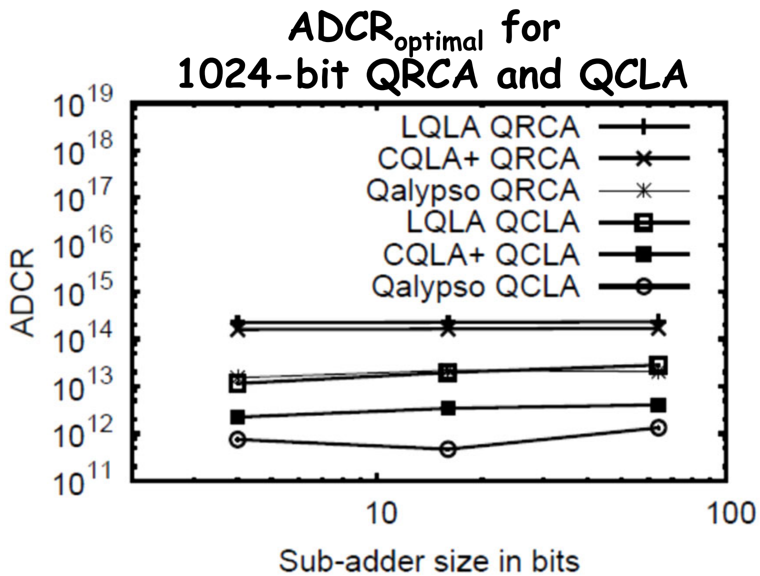


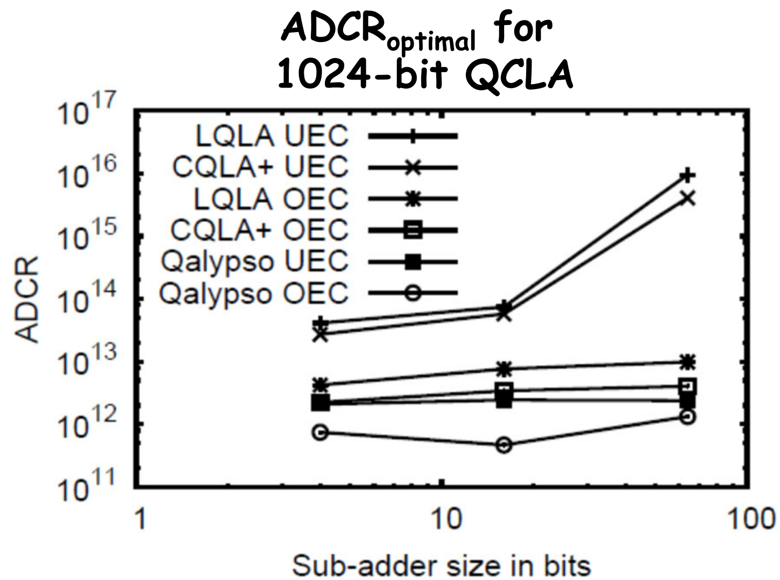
- Then, evaluate probability of success
  - Technique that works well for depolarizing errors: Monte Carlo
    - » Possible error points: Operations, Idle Bits, Communications
  - Vectorized Monte Carlo: n experiments with one pass
  - Need to perform hybrid error analysis for larger circuits
    - » Smaller modules evaluated via vector Monte Carlo
    - » Teleportation infrastructure evaluated via fidelity of EPR bits
- Finally – Compute ADCR for particular result

Quantum CAD flow



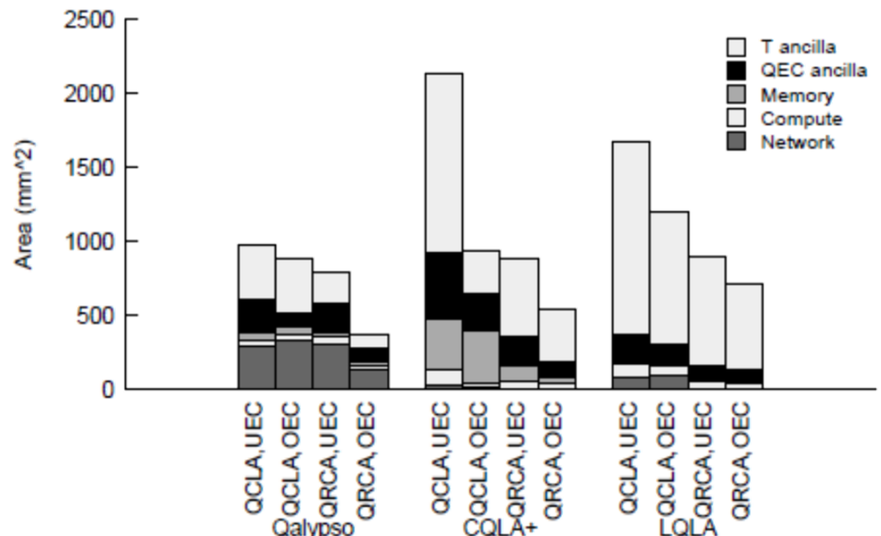
Comparison of 1024-bit adders





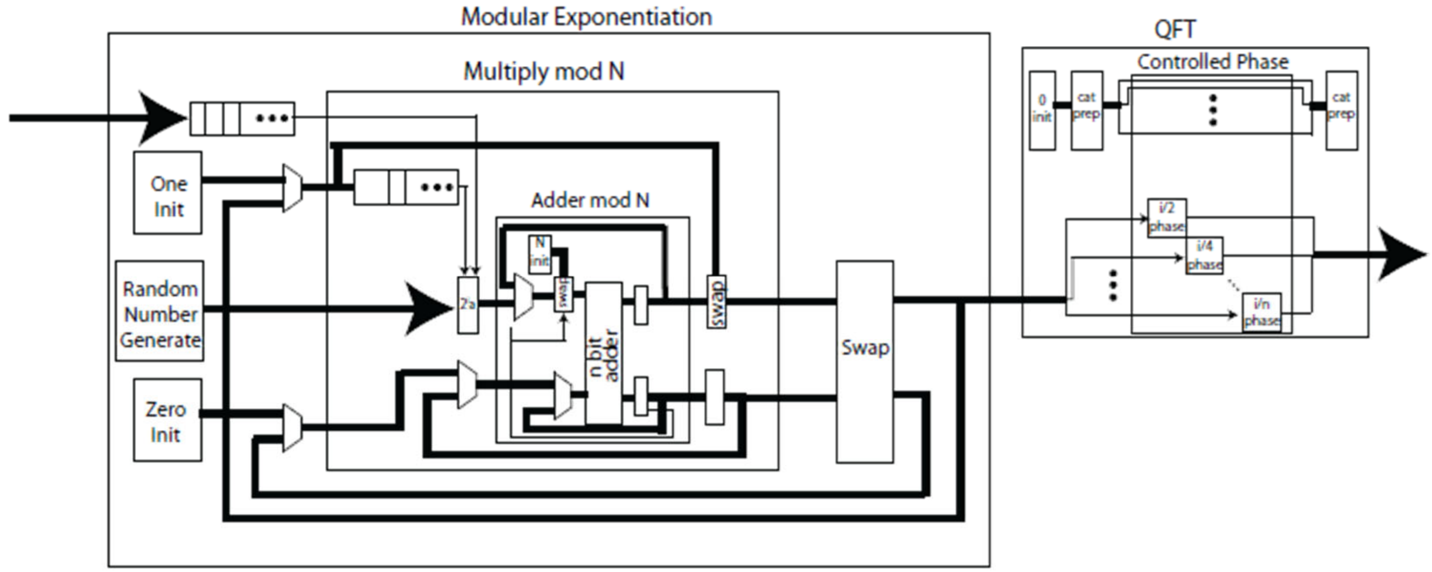
- 1024-bit Quantum Adder Architectures
  - Ripple-Carry (QRCA)
  - Carry-Lookahead (QCLA)
- Carry-Lookahead is better in all architectures
- QEC Optimization improves ADCR by order of magnitude in some circuit configurations

Area Breakdown for Adders



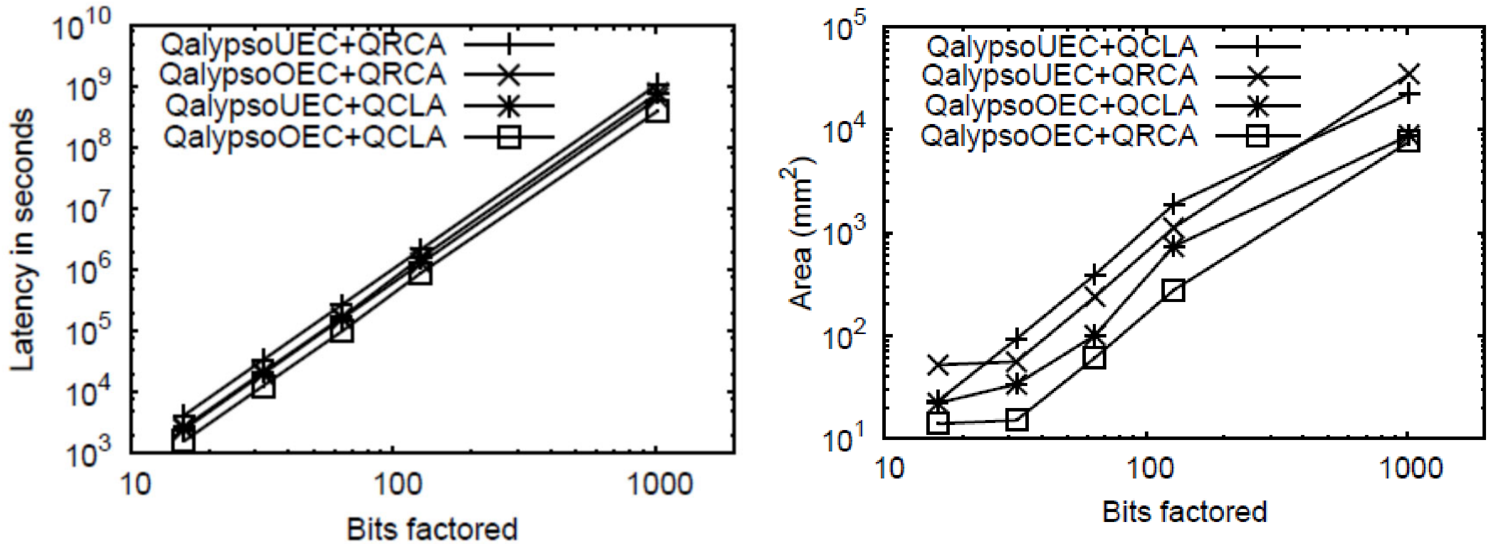
- Error Correction is *not* predominant use of area
  - Only 20-40% of area devoted to QEC ancilla
  - For Optimized Qalypso QCLA, 70% of operations for QEC ancilla generation, but only about 20% of area
- T-Ancilla generation is major component
  - Often overlooked
- Networking is significant portion of area when allowed to optimize for ADCR (30%)
  - CQLA and QLA variants didn't really allow for much flexibility

Investigating 1024-bit Shor's



- Full Layout of all Elements
  - Use of 1024-bit Quantum Adders
  - Optimized error correction
  - Ancilla optimization and Custom Network Layout
- Statistics:
  - Unoptimized version: 1.35×10^{15} operations
  - Optimized Version 1000X smaller
  - QFT is only 1% of total execution time

1024-bit Shor's Continued



- Circuits too big to compute $P_{success}$
  - Working on this problem
- Fastest Circuit: 6×10^8 seconds ~ 19 years
  - Speedup by classically computing recursive squares?
- Smallest Circuit: 7659 mm^2
  - Compare to previous *estimate* of 0.9 m^2 = 9×10^5 mm^2

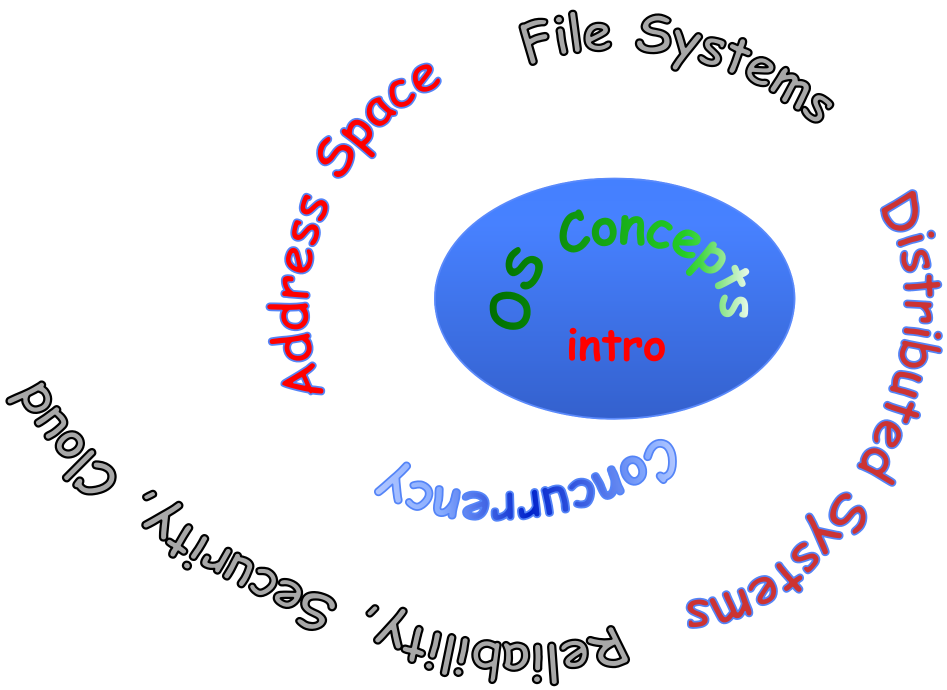
Summary (1/2)

- Remote Procedure Call (RPC): Call procedure on remote machine or in remote domain
  - Provides same interface as procedure
  - Automatic packing and unpacking of arguments without user programming (in stub)
  - Adapts automatically to different hardware and software architectures at remote end
- Key-Value Store:
  - Two operations
    - » put(key, value)
    - » value = get(key)
  - Challenges
    - » Fault Tolerance → replication
    - » Scalability → serve get()'s in parallel; replicate/cache hot tuples
    - » Consistency → quorum consensus to improve put() performance
- Distributed File System:
  - Transparent access to files stored on a remote disk
  - Caching for performance
- Chord:
  - Highly scalable distributed lookup protocol
  - Each node needs to know about O(log(M)), where m is the total number of nodes
  - Guarantees that a tuple is found in O(log(M)) steps
  - Highly resilient: works with high probability even if half of nodes fail

Summary (2/2)

- Quantum Computing
  - Computing using interesting properties of Physics
  - Achieving Quantum Supremacy: Proof that Quantum Computers are more powerful than Classical Ones
    - » Not there yet!
- Most interesting Applications of Quantum Computing:
  - Materials Simulation
  - Optimization problems
  - Machine learning?

Thank you!



- Thanks for all your great questions!
- Good Bye! You have all been great!