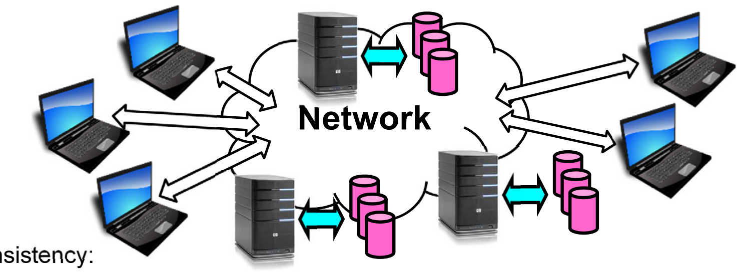
CS162
Operating Systems and Systems Programming
Lecture 26

Key Value Stores (Con't), Chord, DataCapsules
Quantum Computing

December 7th, 2020
Prof. John Kubiatowicz
http://cs162.eecs.Berkeley.edu

Recall: 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"

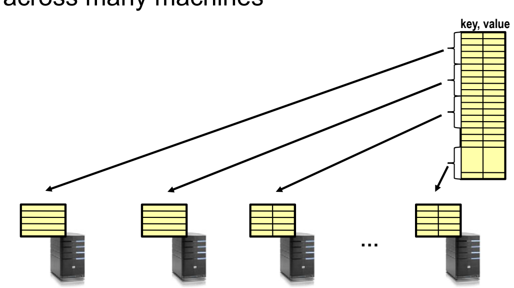
Recall: Key Value Storage

Simple interface

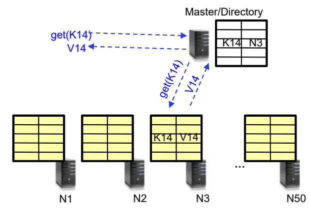
- **put(key, value);** `// Insert/write "value" associated with key`
- **get(key);** `// Retrieve/read value associated with key`

Recall: Key Value Store

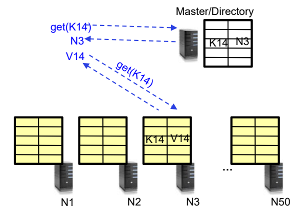
- Also called Distributed Hash Tables (DHT)
- Main idea: simplify storage interface (i.e. put/get), then partition set of key-values across many machines



Recall: Recursive vs. Iterative



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

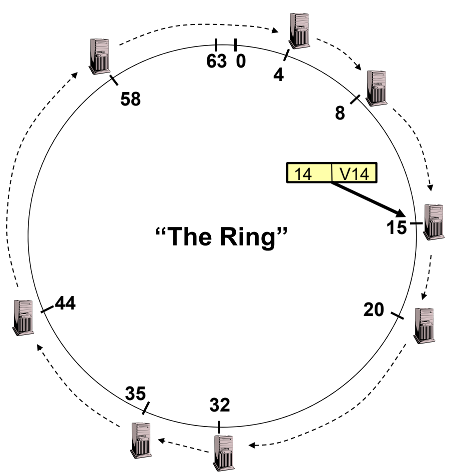
- \+ More scalable, clients do more work
- \- Harder to enforce consistency

Recall: 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 $0..2^m-1$
  ⇒ 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

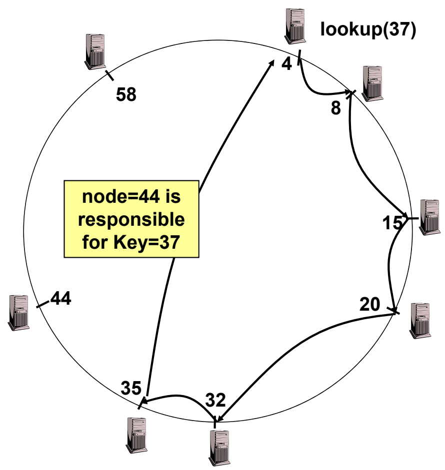


Chord: Distributed Lookup (Directory) Service

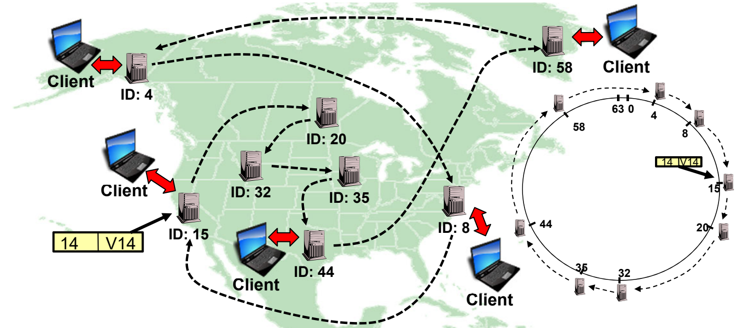
- "Chord" is a Distributed Lookup Service
  - Designed at MIT and here at Berkeley (Ion Stoica among others)
  - Simplest and cleanest algorithm for distributed storage
    - » Serves as comparison point for other optims
- Import aspect of the design space:
  - Decouple correctness from efficiency
  - 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

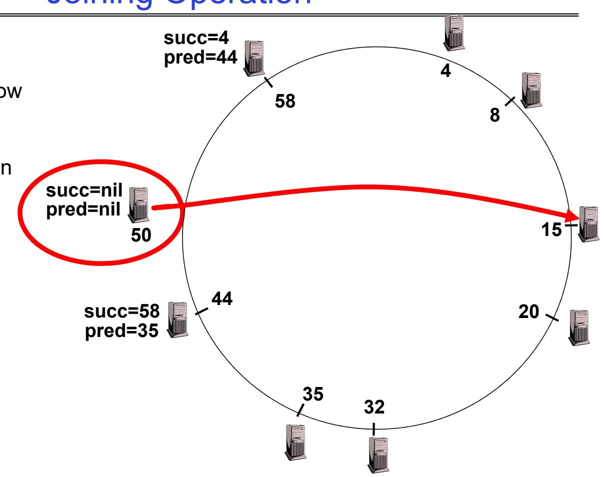
- Periodic operation performed by each node n to maintain its successor when new nodes join the system
  - The primary Correctness constraint

```
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
```

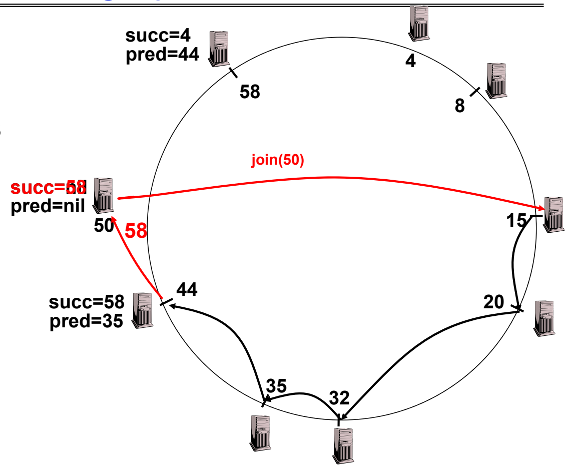

Joining Operation

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



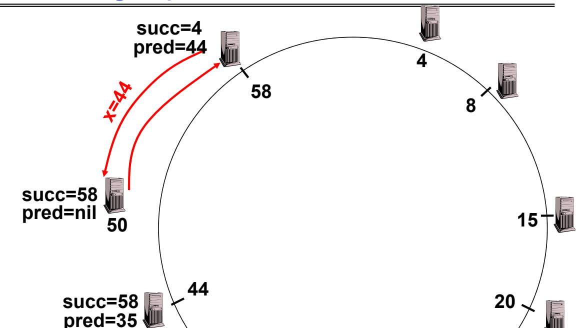
Joining Operation

- 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

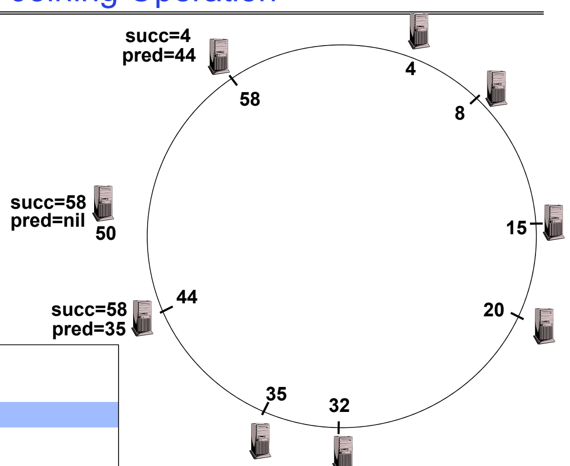
- n=50 executes stabilize()
- n's successor (58) returns x = 44




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


Joining Operation

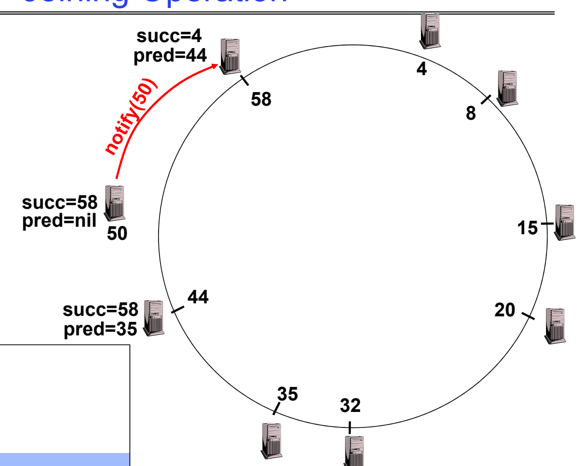
- n=50 executes stabilize()
  - x = 44
  - succ = 58




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


Joining Operation

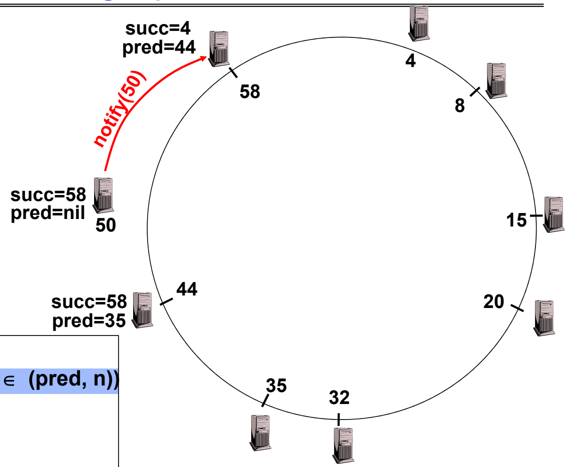
- n=50 executes stabilize()
  - x = 44
  - succ = 58
- n=50 sends to it's successor (58) notify(50)




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

Joining Operation

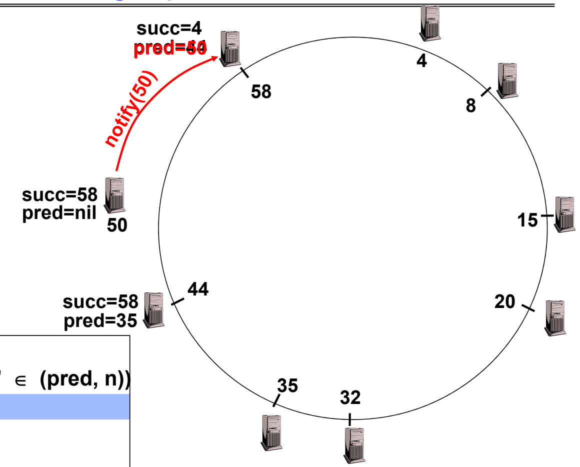
- n=58 executes notify(50)
  - pred = 44
  - n' = 50




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


Joining Operation

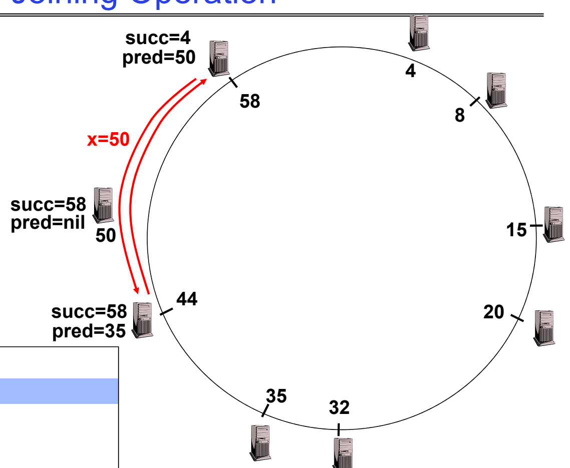
- n=58 executes notify(50)
  - pred = 44
  - n' = 50
- set pred = 50




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


Joining Operation

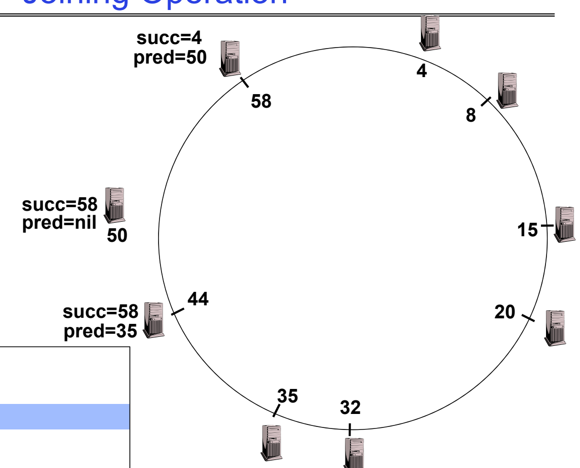
- n=44 executes stabilize()
- n's successor (58) returns x=50




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


Joining Operation

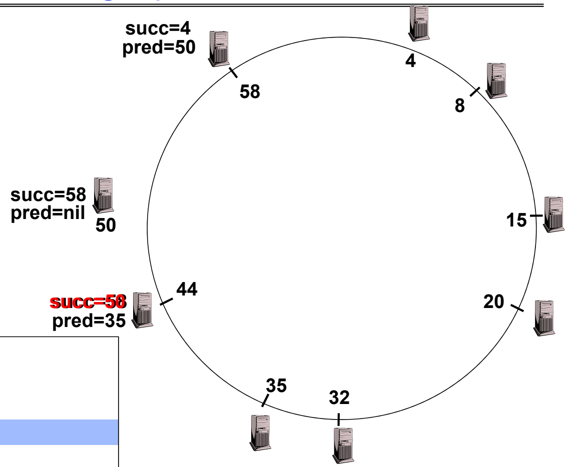
- n=44 executes stabilize()
  - x=50
  - succ=58




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

Joining Operation

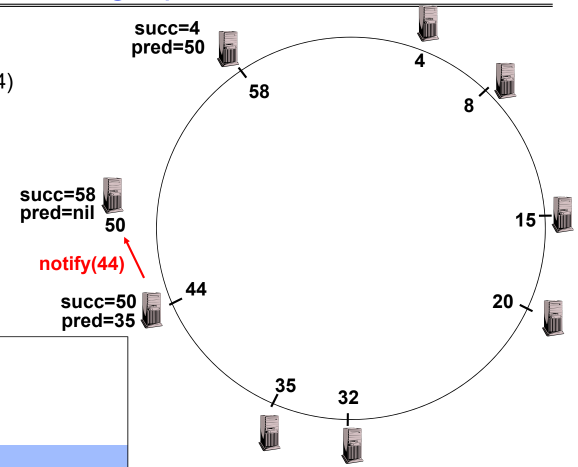
- n=44 executes stabilize()
  - x=50
  - succ=58
- n=44 sets succ=50




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


Joining Operation

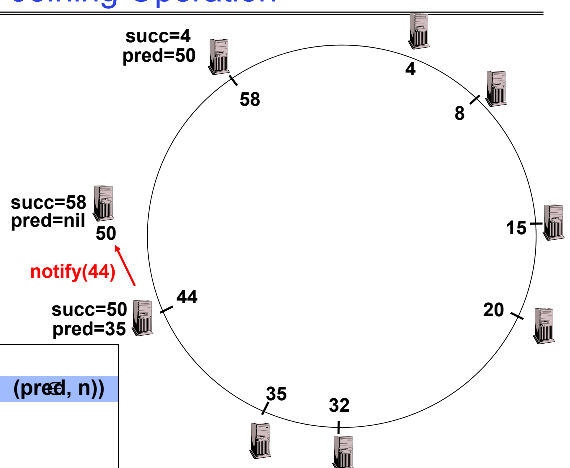
- n=44 executes stabilize()
- n=44 sends notify(44) to its successor




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


Joining Operation

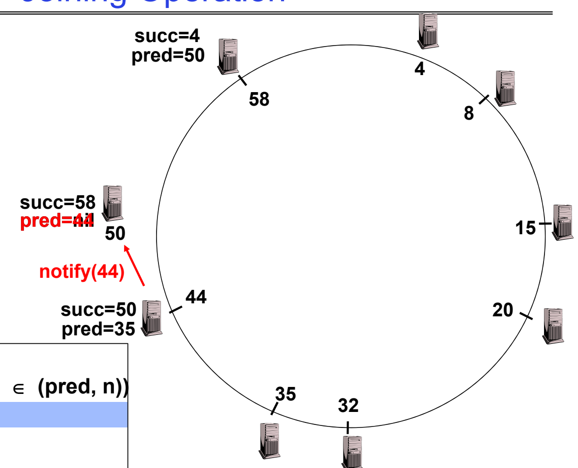
- n=50 executes notify(44)
  - pred=nil




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


Joining Operation

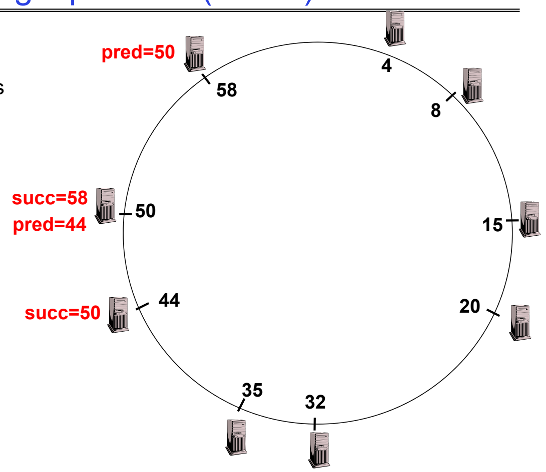
- n=50 executes notify(44)
  - pred=nil
- n=50 sets pred=44




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

Joining Operation (cont'd)

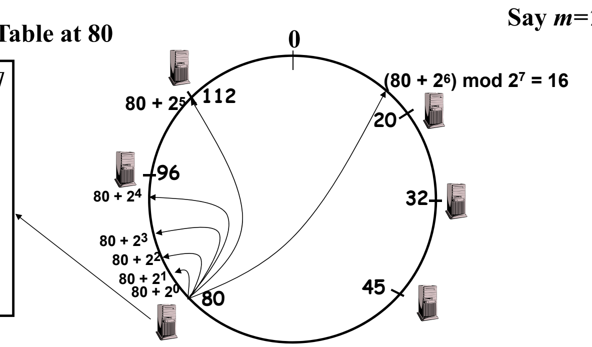
- This completes the joining operation!
- The same stabilizing process will deal with failed nodes by reconnecting the ring
- What if 2 or more nodes in a row fail?
  - Keep track of more neighbors!
  - Called the "leaf set"



Achieving Efficiency: *finger tables*

Finger Table at 80

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



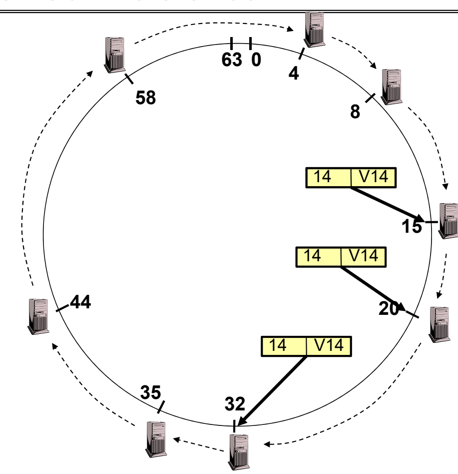
*i*th entry at peer with id *n* is first peer with id >= $n + 2^i \pmod{2^m}$

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

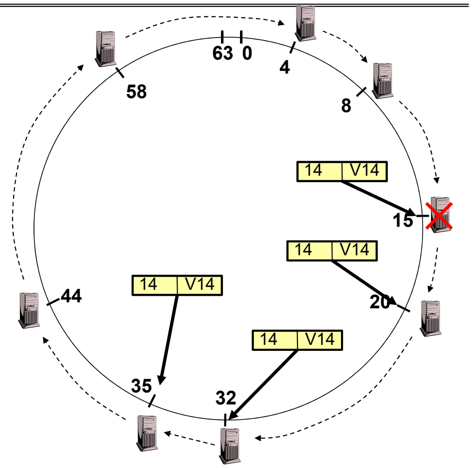
Storage Fault Tolerance

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



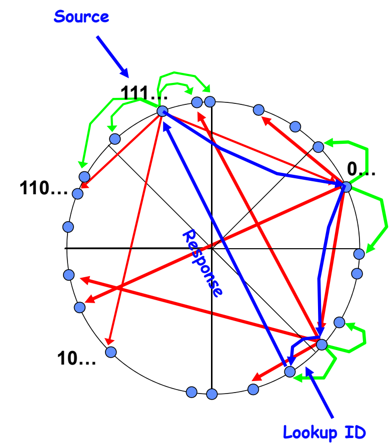
Storage Fault Tolerance

- 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

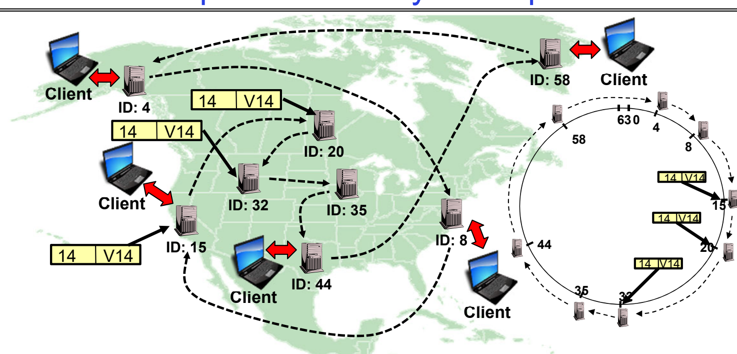


Lookup with Leaf Set

- Assign IDs to nodes
  - Map hash values to node with closest ID
- Leaf set is successors and predecessors
  - All that's needed for correctness
- Routing table matches successively longer prefixes
  - Allows efficient lookups



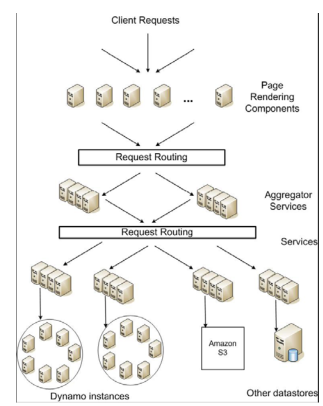
Replication in Physical Space



- 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

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

What is Computer Security Today?

- Computing in the presence of an adversary!
  – Adversary is the security field's defining characteristic
- Reliability, robustness, and fault tolerance
  – Dealing with Mother Nature (random failures)
- Security
  – Dealing with actions of a knowledgeable attacker dedicated to causing harm
  – Surviving malice, and not just mischance
- Wherever there is an adversary, there is a computer security problem!

CIMPLICITY®
BlackEnergy SCADA malware (Supervisory Control and Data Acquisition)

Mirai IoT botnet

Protection vs. Security

- Protection: mechanisms for controlling access of programs, processes, or users to resources
  – Page table mechanism
  – Round-robin schedule
  – Data encryption
- Security: use of protection mechanisms to prevent misuse of resources
  – Misuse defined with respect to policy
    » E.g.: prevent exposure of certain sensitive information
    » E.g.: prevent unauthorized modification/deletion of data
  – Need to consider external operational environment
    » Most well-constructed system cannot protect information if user accidentally reveals password – social engineering challenge

On The Importance of Data Integrity

- In July (2015), a team of researchers took total control of a Jeep SUV remotely
- They exploited a firmware update vulnerability and hijacked the vehicle over the Sprint cellular network
- They could make it speed up, slow down and even veer off the road
- Machine-to-Machine (M2M) communication has reached a dangerous tipping point
  – Cyber Physical Systems use models and behaviors that from elsewhere
  – Firmware, safety protocols, navigation systems, recommendations, …
  – IoT (whatever it is) is everywhere
- Do you know where your data came from? PROVENANCE
- Do you know that it is ordered properly? INTEGRITY
- The rise of Fake Data!
  – Much worse than Fake News…
  – Corrupt the data, make the system behave very badly

Security Requirements

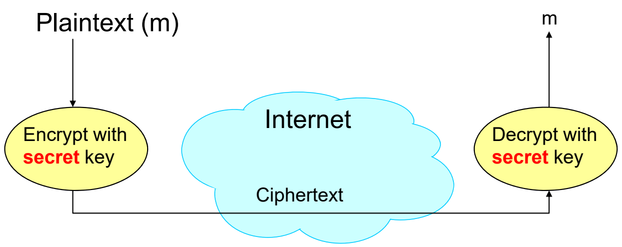
- Authentication
  – Ensures that a user is who they are claiming to be
- Data integrity
  – Ensure that data is not changed from source to destination or after being written on a storage device
- Confidentiality
  – Ensures that data is read only by authorized users
- Non-repudiation
  – Sender/client can't later claim didn't send/write data
  – Receiver/server can't claim didn't receive/write data

Securing Communication: Cryptography

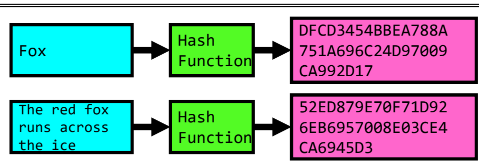
- Cryptography: communication in the presence of adversaries
- Studied for thousands of years
  – See the Simon Singh's The Code Book for an excellent, highly readable history
- Central goal: confidentiality
  – How to encode information so that an adversary can't extract it, but a friend can
- General premise: there is a key, possession of which allows decoding, but without which decoding is infeasible
  – Thus, key must be kept secret and not guessable

Basic Tool: Using Symmetric Keys

- Same key for encryption and decryption
- Achieves confidentiality
- Vulnerable to tampering and replay attacks unless supplement with additional techniques such as nonces
- Good example: AES ("Advanced Encryption Standard")

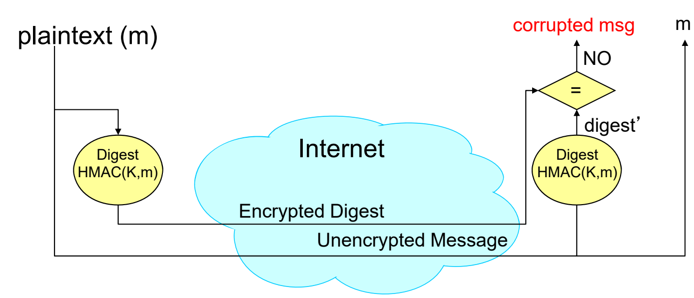


Basic Tool: Secure Hash Function



- Hash Function: Short summary of data (message)
  – For instance, $h_1=H(M_1)$ is the hash of message M_1
    » h_1 fixed length, despite size of message M_1
    » Often, h_1 is called the "digest" of M_1
- Hash function H is considered secure if
  – It is infeasible to find M_2 with $h_1=H(M_2)$; i.e., can't easily find other message with same digest as given message
  – It is infeasible to locate two messages, m_1 and m_2, which "collide", i.e. for which $H(m_1) = H(m_2)$
  – A small change in a message changes many bits of digest/can't tell anything about message given its hash
- Good example: SHA-256

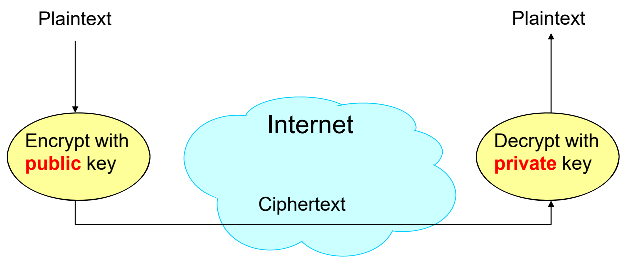
Using Hashing for Integrity



Can encrypt m for confidentiality

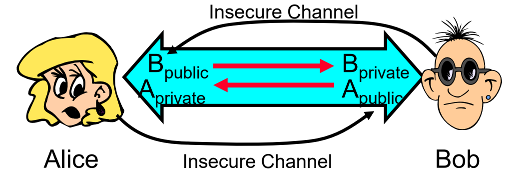
Basic Tool: Public Key / Asymmetric Encryption

- Instead of one key, have two keys: public and private
- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



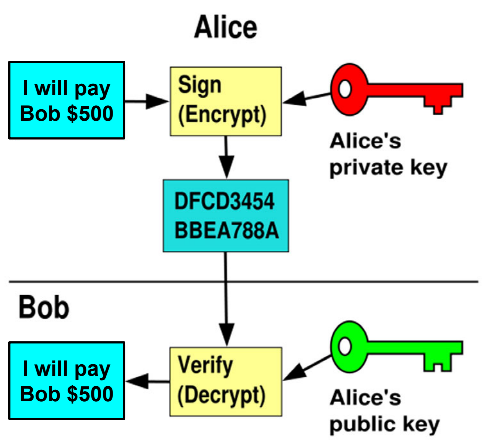
Public Key Encryption Details

- Idea: K_{public} can be made public, keep $K_{private}$ private



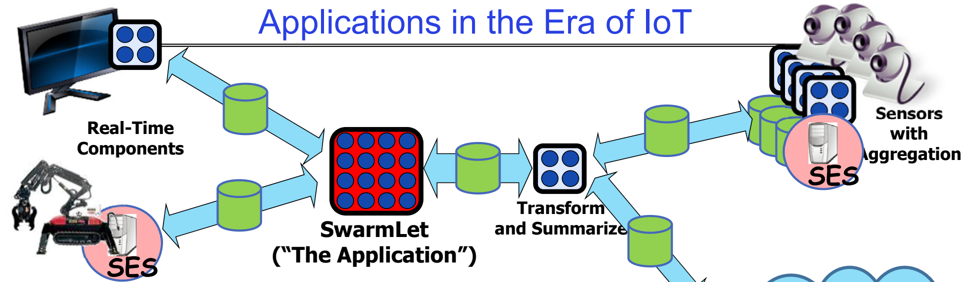
- Gives message privacy (restricted receiver):
  - Public keys (secure destination points) can be acquired by anyone/used by anyone
  - Only person with private key can decrypt message
- What about authentication?
  - Use combination of private and public key
  - Alice→Bob: $[(\text{I'm Alice})^{A_{private}} \text{ Rest of message}]^{B_{public}}$
  - Provides restricted sender and receiver
- But: how does Alice know it was Bob who sent her B_{public}? And vice versa…
  - Need a key distribution mechanism/Public Key Infrastructure

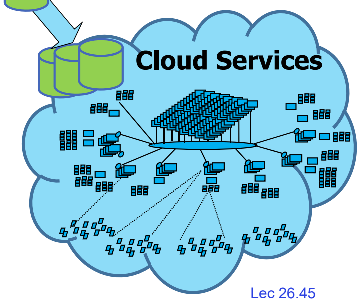
Public Key Crypto & Signatures



Fog Robotics and the Global Data Plane (GDP)

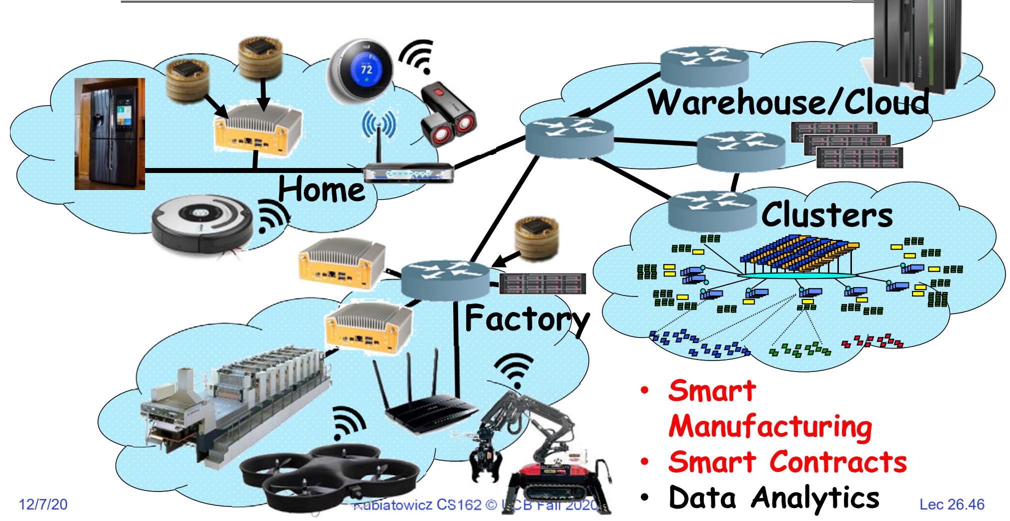
Applications in the Era of IoT





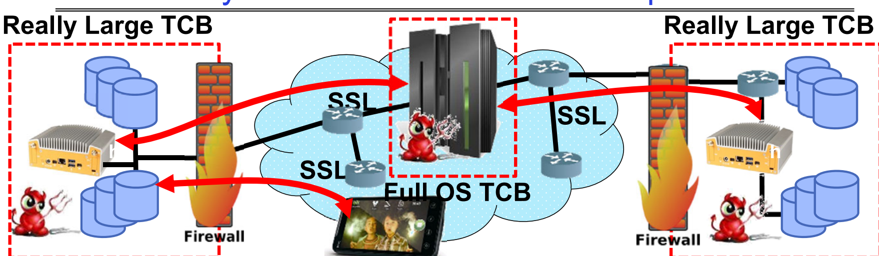
- An Application is a Connected Graph of Services
  - Locality and QoS aware!
  - Use local resources to limit external observability/interference
- Distributed storage everywhere
  - Each arrow represents imbedded storage
  - Transient or Long term
- Computation on the edge of the network
  - Perhaps Secure Enclaves (SES) for trusted computation…?
  - Rapid launching of computation to close resources

A Physical View of these Applications: Distributed, Ad Hoc, and Vulnerable



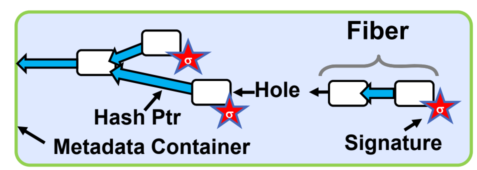
- Smart Manufacturing
- Smart Contracts
- Data Analytics

Why are Data Breaches so Frequent?



- State of the art: AdHoc boundary construction!
  - Protection mechanisms are all "roll-your-own" and different for each application
  - Use of encrypted channels to "tunnel" across untrusted domains
- Data is protected at the *Border* rather than *Inherently*
  - Large Trusted Computing Base (TCB): huge amount of code must be correct to protect data
  - Make it through the border (firewall, OS, VM, container, etc…) data compromised!
- What about data integrity and provenance?
  - Any bits inserted into "secure" environment get trusted as authentic ⇒ manufacturing faults or human injury or exposure of sensitive information

The Data-Centric Vision: Cryptographically Hardened Data Containers

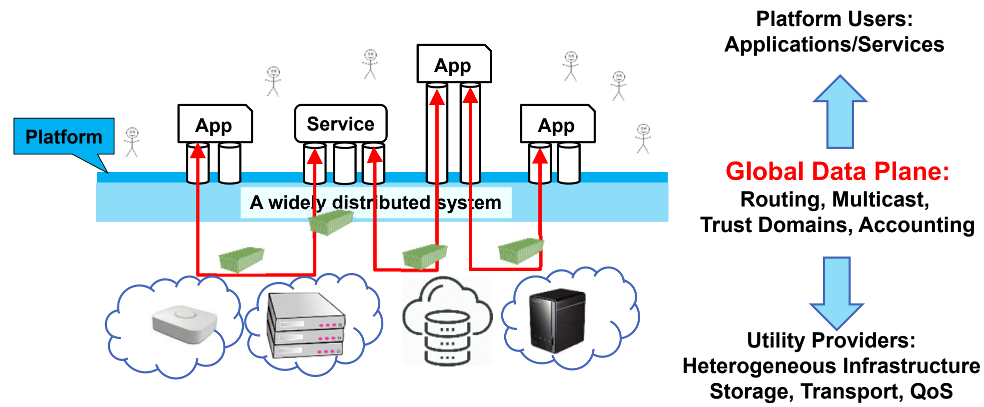


- Inspiration: Shipping Containers
  - Invented in 1956. Changed everything!
  - Ships, trains, trucks, cranes handle *standardized format containers*
  - *Each container has a unique ID*
  - *Can ship (and store) anything*
- *Can we use this idea to help?*
- DataCapsule (DC):
  - Standardized metadata wrapped around opaque data transactions
  - Uniquely named and globally findable
  - Every transaction explicitly sequenced in a hash-chain history
  - Provenance enforced through signatures
- Underlying infrastructure assists and improves performance
  - Anyone can verify validity, membership, and sequencing of transactions (like blockchain)

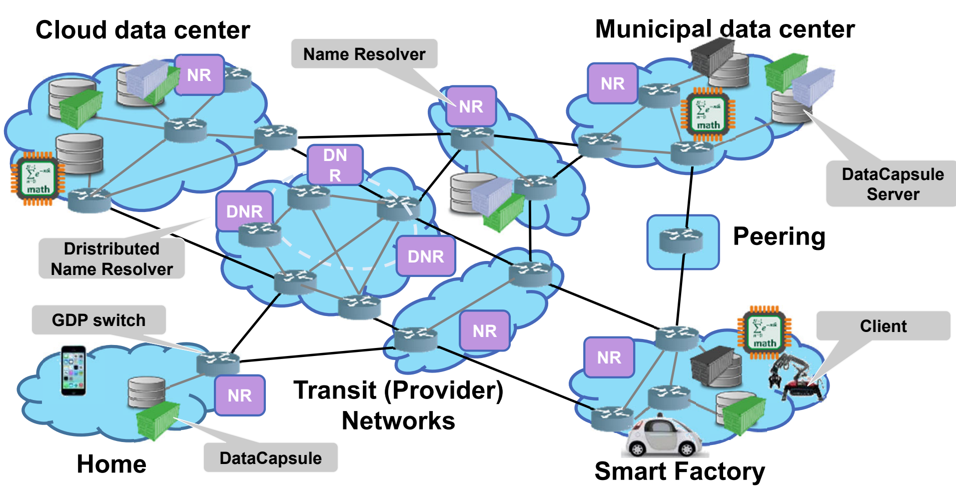
Why does this help?

- The "Networking" effect (Pun Intended!)
  – Standardization ⇒ Infrastructure proliferation that benefits everyone
  – Federation ⇒ Enable a market of service providers
- Data becomes a first-class entity in the network!
  – Asserts its own requirements for security, privacy, which are enforced via cryptography
  – Independent of physical location – policies can target durability, QoS, availability, etc
  – No application silos – data producers own and chose how to share their information
  – Network is informed about the information that it is carrying and where it may go
- First (Necessary) Step: Network Cannot Enforce what is not Specified!
- Related information bundled and kept together as it migrates
  – Provenance and data ordering part of all information usage
  – Information labeled with meta-data about (1) Where it is allowed to be in the network, and (2) Who is allowed to view and interact with it, (3) Who is allowed to modify it.

A Platform Approach: the Utility-Provider Model [DataCapsule version of Ships, Trains, Trucks, and Cranes]

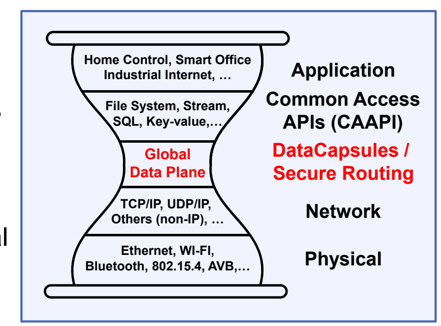


A Physical View of the GDP

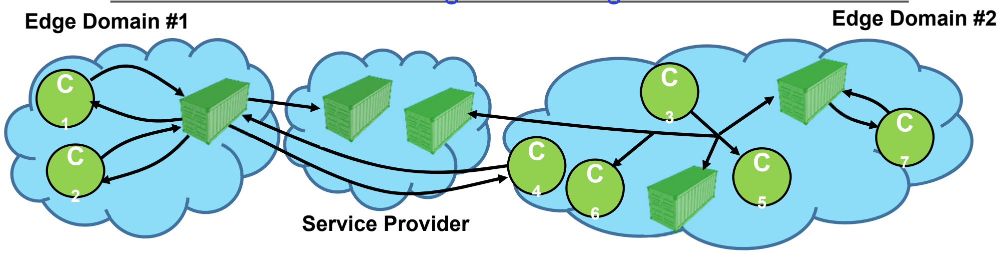


Refactoring of Applications around Security, Integrity, and Provenance of Information

- Goal: A thin Standardized entity that can be easily adopted and have immediate impact
  – Can be embedded in edge environments
  – Can be exploited in the cloud
  – Natural adjunct to Secure Enclaves for computation
- DataCapsules ⇒ bottom-half of a blockchain?
  – Or a GIT-style version history
  – Simplest mode: a secure log of information
  – Universal unique name ⇒ permanent reference
- Applications writers think in terms of traditional storage access patterns:
  – File Systems, Data Bases, Key-Value stores
  – Called Common Access APIs (CAAPIs)
  – DataCapsules are always the *Ground Truth*

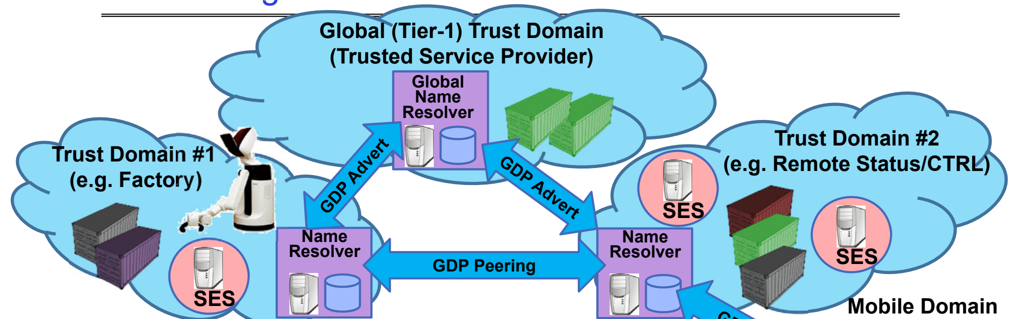


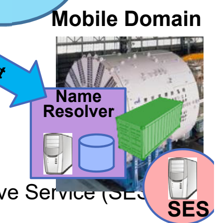
Global Data Plane (GDP) and the Secure Datagram Routing Protocol



- Flat Address Space Routing
  - Route queries to DCs by names, independent of location (e.g. no IP)
  - DCs move, network deals with it
  - Short-term Channels ("µ-SSL channels")
- Black Hole Elimination: Delegation of Names
  - Only servers authorized by owner of DC may advertise DC service
- Routing only through domains you trust!
  - Secure Delegated Flat Address Routing
- Secure Multicast Protocol
  - Only clients/DC storage servers with proper (delegation) certificates may join
- Queries (messages) are *Fibers*
  - Self-verifying chunks of DataCapsules
  - Writes include appropriate credentials
  - Reads include proofs of membership
- Incremental deployment as an overlay
  - Prototype tunneling protocol ("GDPinUDP")
  - Federated infrastructure w/routing certificates

Reasoning about the infrastructure: Trust Domains



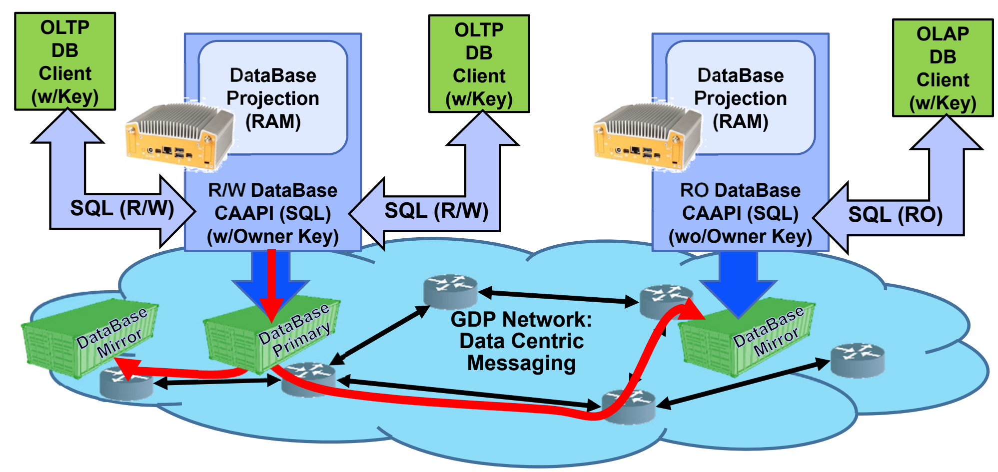


- Trust Domains: Groups of Resources owned by single entity
  - Reflect the ownership, trustworthiness, and degree of maintence
  - Carry unique economic, political, or incentive structure of the owner
  - Pay-for-service, Federated utility model
- Trust for:
  - Message Transport, Location Resolution, DataCapsule Service, Secure Enclave Service (SES)
  - Conversations routed according to DataCapsule owner's Trust Preferences

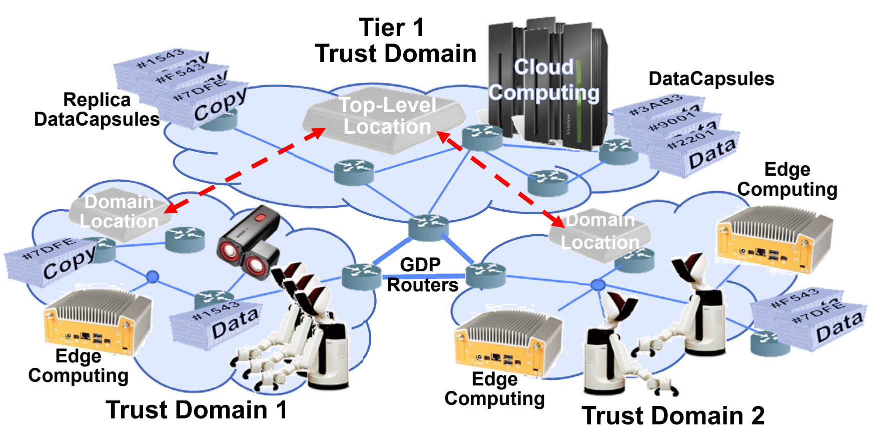
Common Access APIs (CAAPIs)

- Common Access APIs (CAAPIs) provide convenient/familiar *Storage Access Patterns*:
  - Random File access, Indexing, SQL queries, Latest value for given Key, etc
  - Optional Checkpoints for quick restart/cloning
  - Refactoring: CAAPIs are services or libraries running in trusted or secured computing environments on top of DataCapsule infrastructure
- Many Consistency Models possible
  - DataCapsules are "Conflict-free Replicated Data Types" (CRDTs): Synchronization via Union
  - Single-Writer CAAPIs prevent branches if sufficient stable storage (strong consistency models)
  - DataCapsules with branches: like GIT or Amazon Dynamo (write always, reader handles branches)
  - CAAPIs can support anything from weak consistency to serializability
- Examples:
  - Streaming storage
  - Key/Value store with time-travel
  - Filesystem (changeable sequences of bytes organized in hierarchy)
  - Multi-writer storage using Paxos or RAFT
  - Byzantine agreement with threshold admission to DataCapsules

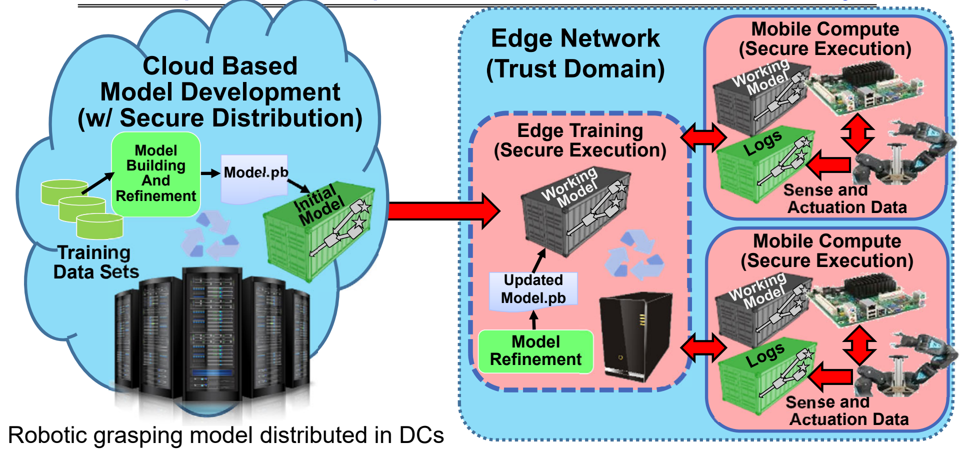
E.g. Using DataCapsules to support more familiar data access patterns (e.g. DataBase)



Fog Robotics on the Global Data Plane

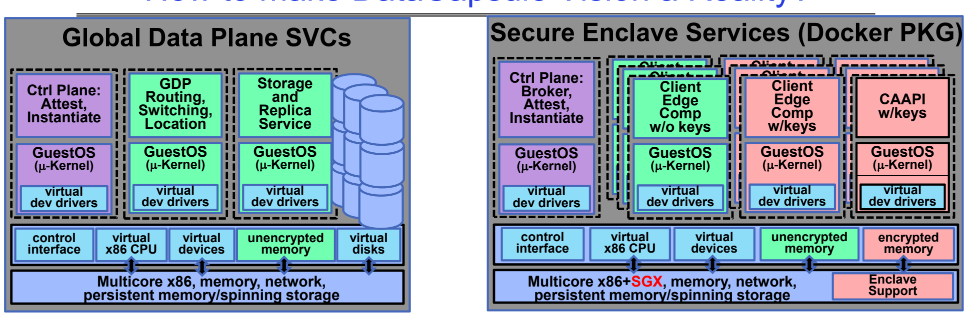


Example: Data Capsules as Part of Model Delivery



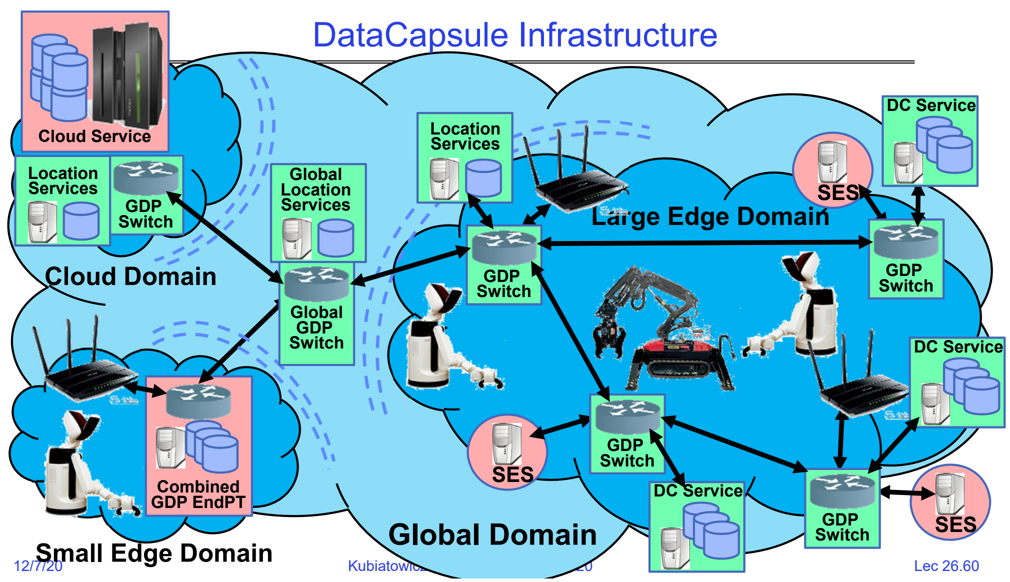
- Robotic grasping model distributed in DCs
  - Intellectual property of producer (only unpacked in environments guaranteed not to leak model)
  - Refinement on the edge is updated only by authorized enclaves with attested algorithms

How to make DataCapsule Vision a Reality?



- Active Routing/Switching Components
  - Federated/Utility storage infrastructure
  - Edge-local support for multicast
  - Data Location Services
- Owned by service provider (trust domain)
  - Secure boot/validated code in DataCapsule
  - Multiple providers may own equipment in single physical environment
- Multi-Tenant Secure Computation Services
  - Secure Enclaves on Demand with specified attributes (e.g. GPU, special accelerator, etc.)
  - Standardized packaging (e.g. Docket)
  - Trustable computation through attestation, key exchange, resistance to physical attacks
- Computation is *fungible:*
  - Executable and state stored in DataCapsules!

DataCapsule Infrastructure



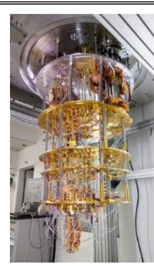
Quantum Computing, Shor's Algorithm, and the role of CAD design

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

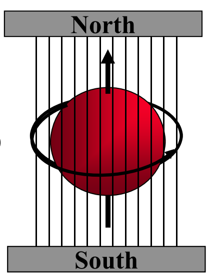


IBM: Superconducting Devices up to 50 qubits

- 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"

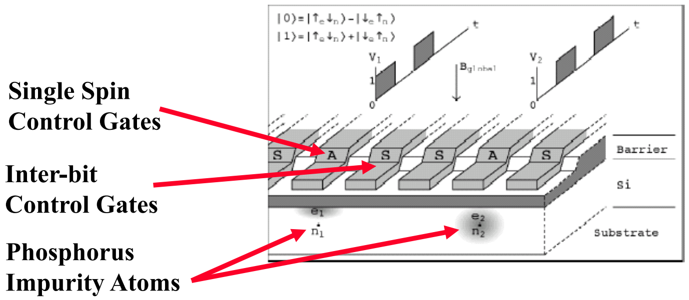
Spin ½ particle: (Proton/Electron)



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)



- 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> (say "UP")
  - With probability $|C_1|^2$ we will find |1> (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> 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!

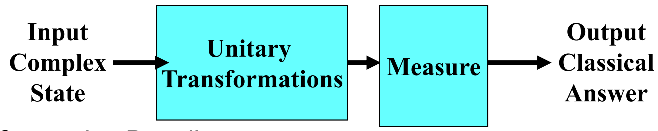
Spooky action at a distance

- 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

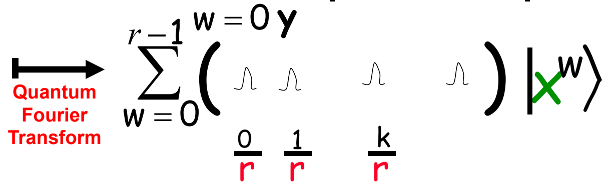
- The Security of RSA Public-key cryptosystems depends on the difficulty of factoring a number N=pq (product of two primes)
  - Classical computer: sub-exponential time factoring
  - Quantum computer: polynomial time factoring
- Shor's Factoring Algorithm (for a quantum computer)

Easy 1) Choose random $x : 2 \le x \le N-1$.
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, GOTO 1
Easy 5) If r is even, $a \equiv x^{r/2} \pmod N \Rightarrow (a-1)\times(a+1) = kN$
Easy 6) If $a \equiv N-1 \pmod N$ GOTO 1
Easy 7) ELSE gcd($a \pm 1,N$) is a non trivial factor of N.

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

$$\sum_k |k\rangle|1\rangle \longrightarrow \sum_k |k\rangle|x^k\rangle$$

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

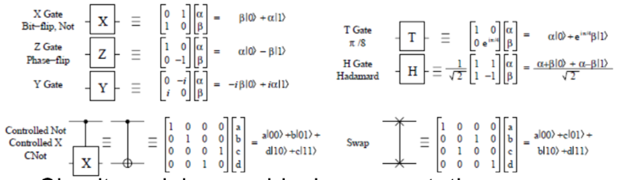


- Finally: Perform measurement
  - Find out r with high probability
  - Get |y>|a^{w'}> where y is of form k/r and w' is related

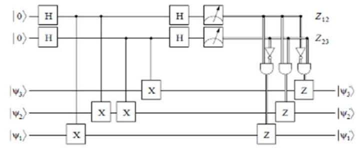
Quantum Computing Architectures

- 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

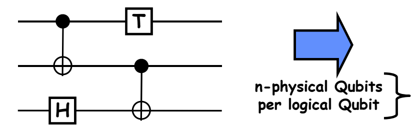
Quantum Circuit Model

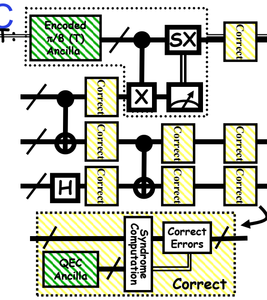


- Quantum Circuit model – graphical representation
  - Time Flows from left to right
  - Single Wires: persistent Qubits, Double Wires: classical bits
    - » Qubit – coherent combination of 0 and 1: $\psi = \alpha|0\rangle + \beta|1\rangle$
  - Universal gate set: Sufficient to form all unitary transformations
- 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





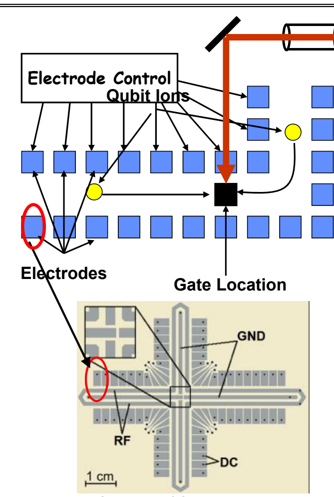
- Quantum State Fragile ⇒ encode all Qubits
  - Uses many resources: e.g. 3-level [[7,1,3]] code 343 physical Qubits/logical Qubit)!
- Still need to handle operations (fault-tolerantly)
  - *Some* set of gates are simply "transversal:"
    - » Perform identical gate between each physical bit of logical 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

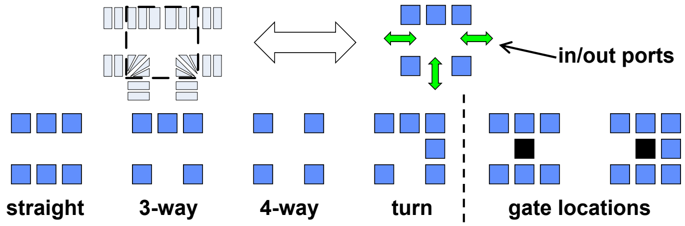
- Qubits are atomic ions (e.g. Be^+)
  - State is stored in hyperfine levels
  - Ions suspended in channels between electrodes
- Quantum gates performed by lasers (either one or two bit ops)
  - Only at certain trap locations
  - Ions move between laser sites to perform gates
- Classical control
  - 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

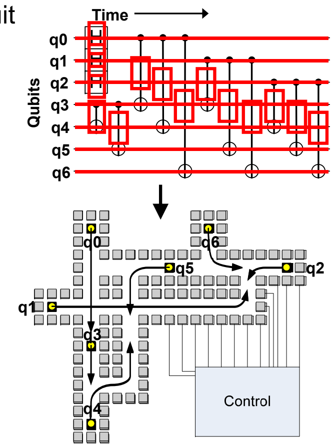
- *Basic block* abstraction: Simplify Layout



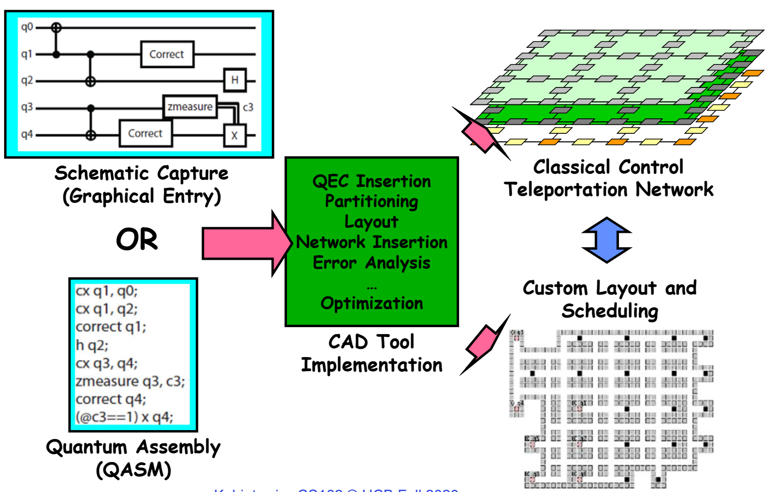
- Evaluation of layout through simulation
  - Yields Computation Time and Probability of Success
- Simple Error Model: Depolarizing Errors
  - Errors for every Gate Operation and Unit of Waiting
  - Ballistic Movement Error: Two error Models
    1. Every Hop/Turn has probability of error
    2. Only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

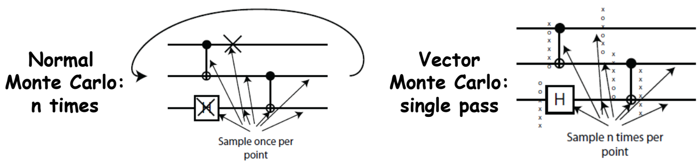


Important Measurement Metrics

- 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 $\times$ 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

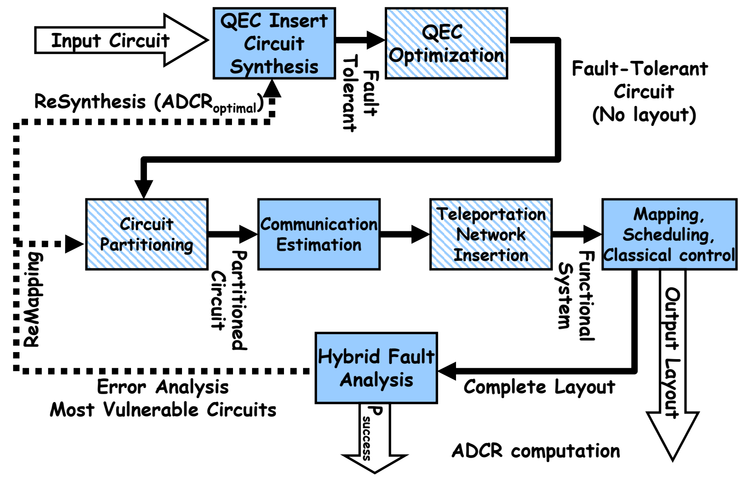
How to evaluate a circuit?

- 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

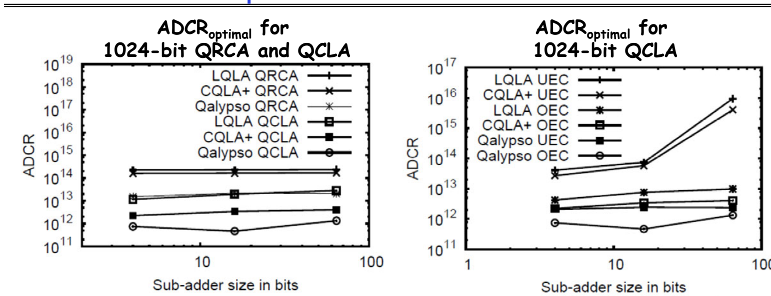


- Then, evaluate probability of success
  - Technique that works well for depolarizing errors: Monte Carlo
    - » Possible error points: Operations, Idle Bits, Communications
  - Vectorized Monte Carlo: n experiments with one pass
  - Need to perform hybrid error analysis for larger circuits
    - »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

Quantum CAD flow

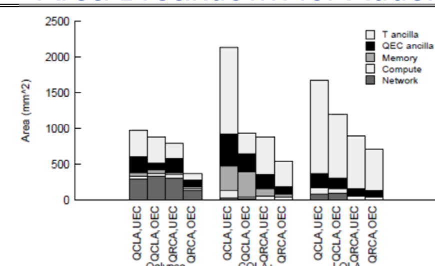


Comparison of 1024-bit adders



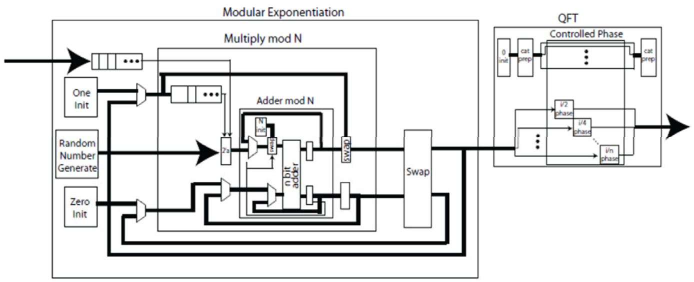
- 1024-bit Quantum Adder Architectures
  - Ripple-Carry (QRCA)
  - Carry-Lookahead (QCLA)
- Carry-Lookahead is better in all architectures
- QEC Optimization improves ADCR by order of magnitude in some circuit configurations

Area Breakdown for Adders



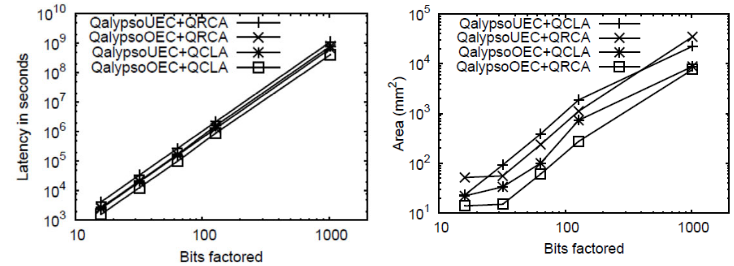
- Error Correction is *not* predominant use of area
  - Only 20-40% of area devoted to QEC ancilla
  - For Optimized Qalypso QCLA, 70% of operations for QEC ancilla generation, but only about 20% of area
- T-Ancilla generation is major component
  - Often overlooked
- Networking is significant portion 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

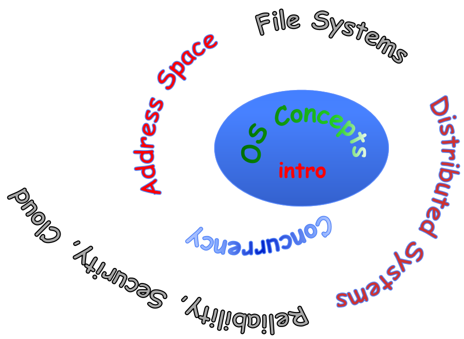
Summary (1/2)

- 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
- 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)

- Cryptography is a mechanism that is helpful for enforcing a security policy
  - Encryption, Hashing, Digital Signatures
- It's all about the Data!
  - Hardening the Data while freeing it to reside anywhere
  - Edge Computing Enabled by DataCapsules
- 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!