Distributed Systems

15-440/640

Fall 2018

Review

Midterm II on Dec 6 in the McConomy Auditorium

Midterm II on 11/29

15) Data Center Storage: GFS / HDFS

~~19) Guest Lecture~~

16 & 17) Cluster Computing: MapReduce/Hadoop

Spark & Distributed ML

18) Internet Content Delivery: DNS and CDNs

20) Virtualization Technology: VMs & Containers

21) Byzantine Fault Tolerance

22) Distributed ledgers and Blockchains

23 & 24) Security: Protocols

Layering and Security

Distributed Systems

15-440/640

Fall 2018

15 – Cluster File Systems: The Google File System

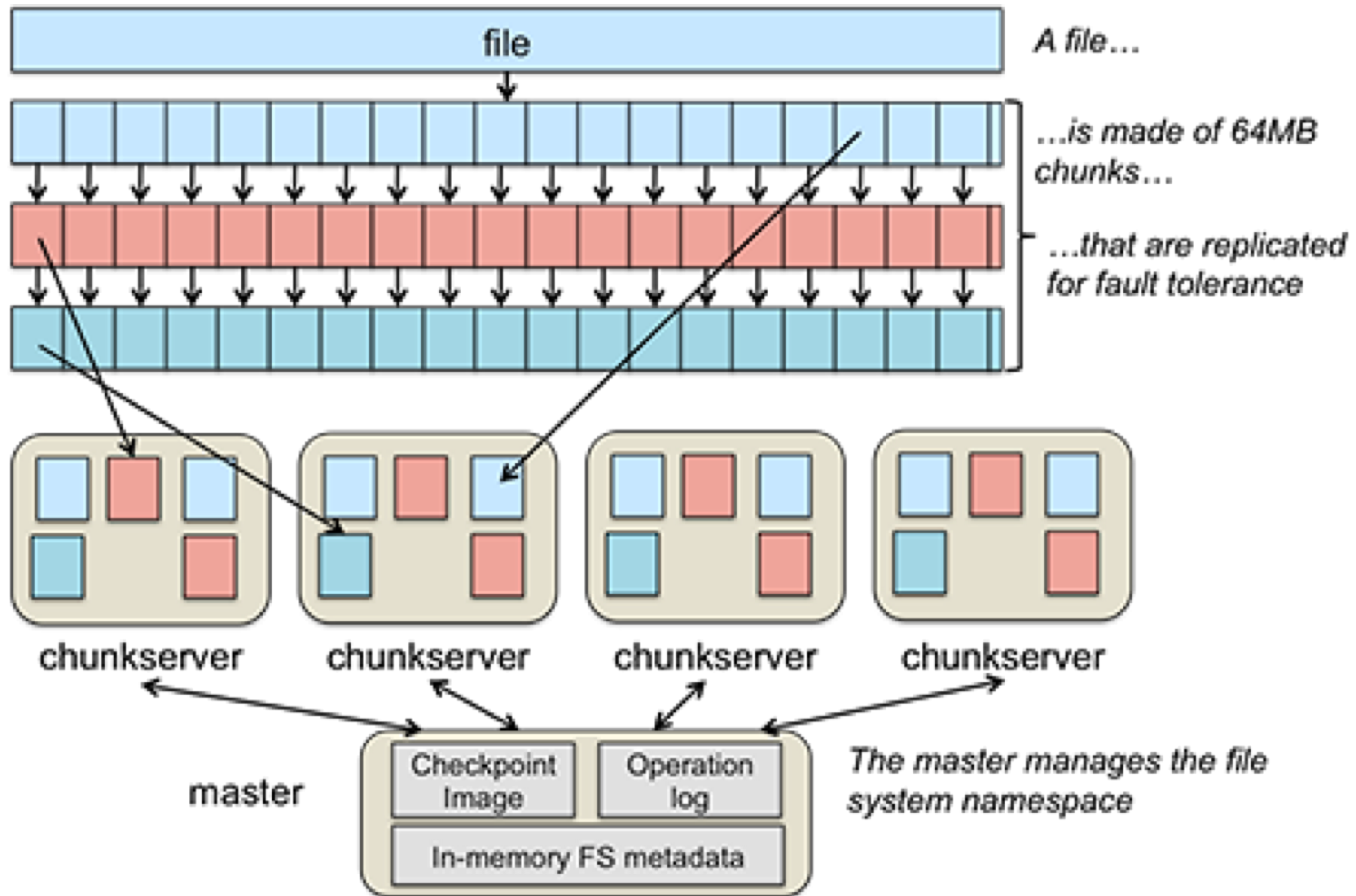
Readings: “The Google File System” Sections 2.3-2.6, 3.1, 3.3, 5.1, 5.2

Daniel S. Berger

GFS: Workload Assumptions

- Large files, >= 100 MB in size
- Large, streaming reads (>= 1 MB in size)
  - Read once
- Large, sequential writes that append
  - Write once
- Concurrent appends by multiple clients (e.g., producer-consumer queues)
  - Want atomicity for appends without synchronization overhead among clients

Master/Chunkservers



GFS Architecture

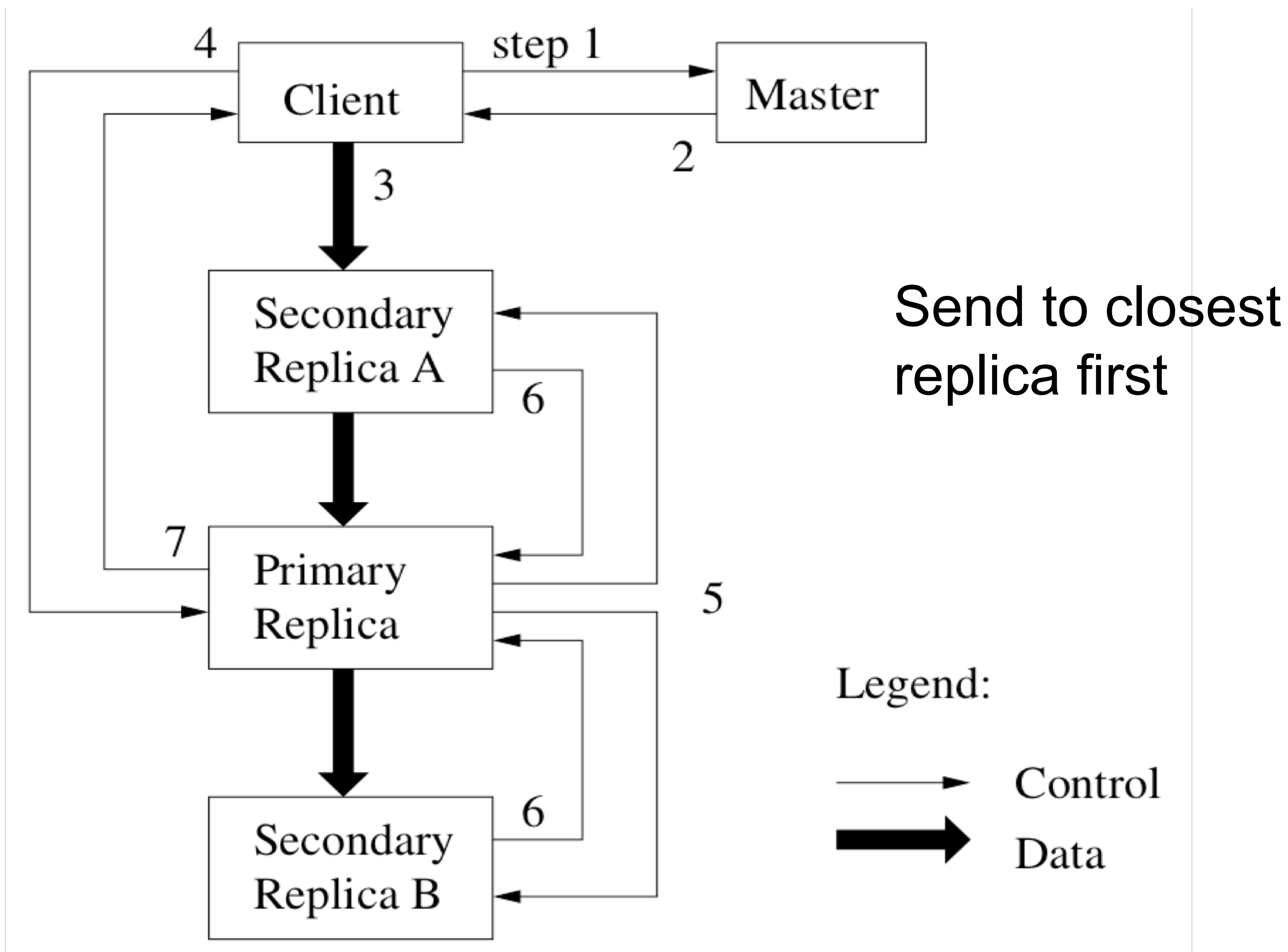
Application
GFS client
(file name, chunk index)
(chunk handle,
chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system

Legend:

Data messages

Control messages

GFS Client Write Operation III

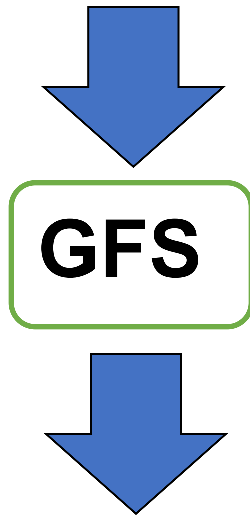


Send to closest replica first

GFS Record Append Operation

- Google uses large files as queues between multiple producers and consumers
- Variant of GFS write step

Why not use a regular GFS write (client offset)?



- Client pushes data to last chunk's replicas
- Client sends request to primary
- Common case: request fits in last chunk:
  - Primary appends data to own chunk replica
  - Primary tells secondaries to do same at same byte offset in their chunk replicas
  - Primary replies with success to client

GFS Append if Chunk is Full

- When data won't fit in last chunk:
  - Primary fills current chunk with padding
  - Primary instructs other replicas to do same
  - Primary replies to client, "retry on next chunk"
- If record append fails at any replica, client retries operation

What guarantee does GFS provide after reporting success of append to application?

- Replicas of same chunk may contain different data—even duplicates of all or part of record data
- Data written **at least once** in atomic unit

  ⇒ due to GFS client retries until success

GFS Consistency Model (Data)

- Changes to data are ordered as chosen by a primary
  - But multiple writes from the same client may be interleaved or overwritten by concurrent operations from other clients
- Record append completes at least once, at offset of GFS's choosing
  - Applications must cope with possible duplicates
- Failures can cause inconsistency
  - E.g., different data across chunk servers (failed append)
  - Behavior is worse for writes than appends

GFS Limitations

- Does not mask all forms of data corruption
  - Requires application-level checksum
- Master biggest impediment to scaling
  - Performance and availability bottleneck
  - Takes long time to rebuild metadata
  - Solution:
    - Multiple master nodes, all sharing set of chunk servers. Not a uniform name space.
- Large chunk size
  - Can't afford to make smaller
- Security?
  - Trusted environment, but users can interfere

Distributed Systems

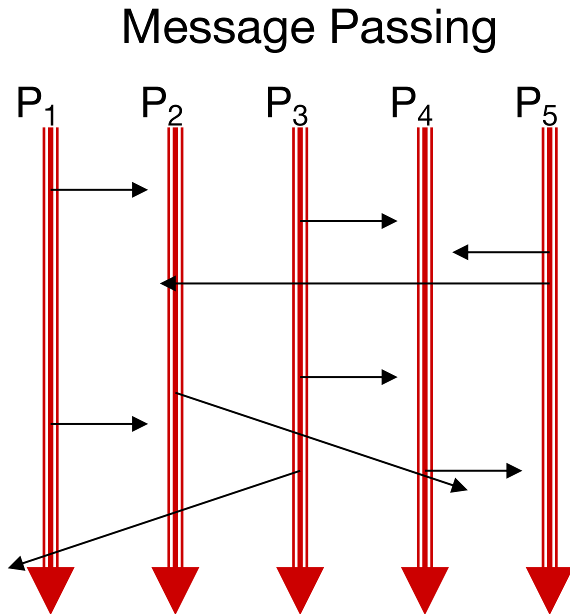
15-440/640

Fall 2018

16 – Cluster Computing: MPI & MapReduce

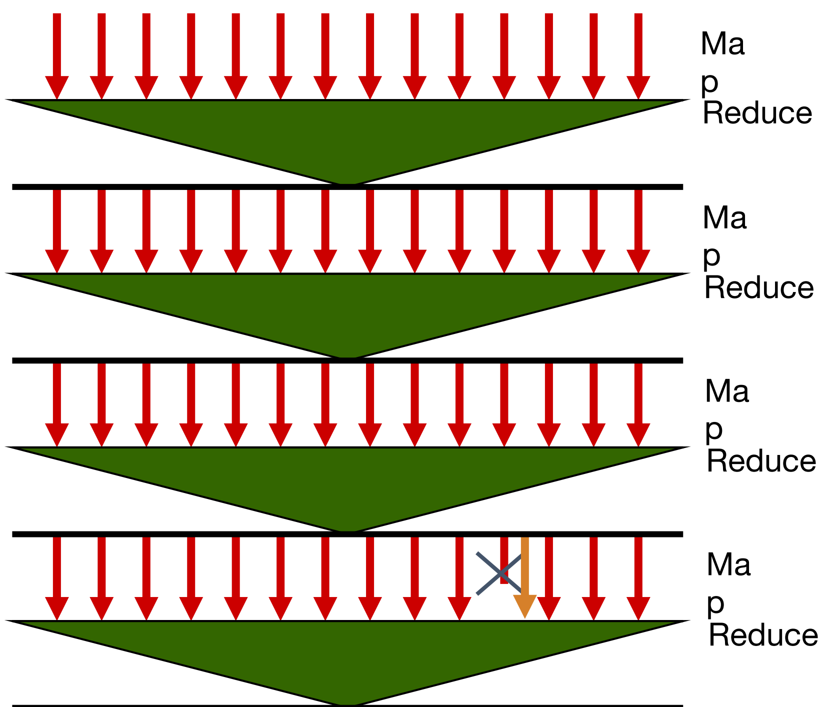
Readings: "MapReduce: Simplified Data Processing on Large Clusters" Sections 3,4

Typical HPC Operation



- Characteristics
  - Long-lived processes
  - Partitioning: exploit spatial locality
  - Hold all program data in memory (no disk access)
  - High bandwidth communication
- Strengths
  - High utilization of resources
  - Effective for many scientific applications
- Weaknesses
  - Requires careful tuning of application to resources
  - Intolerant of any variability

Map/Reduce Operation

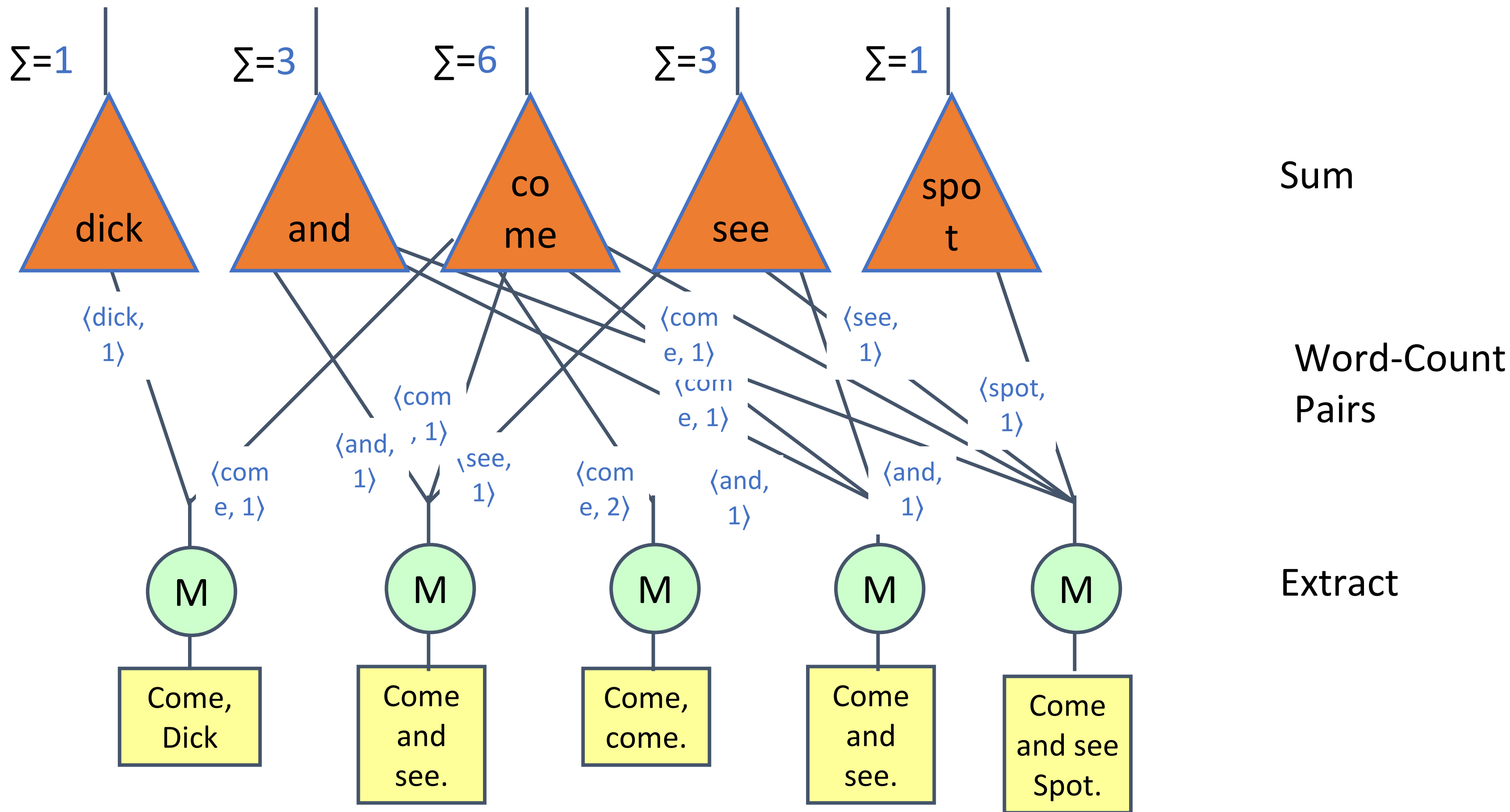


- Characteristics
  - Computation broken into many, short-lived tasks
  - Use disk storage to hold intermediate results
  - Failure → Reschedule task
- Strengths
  - Great flexibility in placement, scheduling, and load balancing
  - Can access large data sets
- Weaknesses
  - Higher overhead
  - Lower raw performance

Hadoop MapReduce API

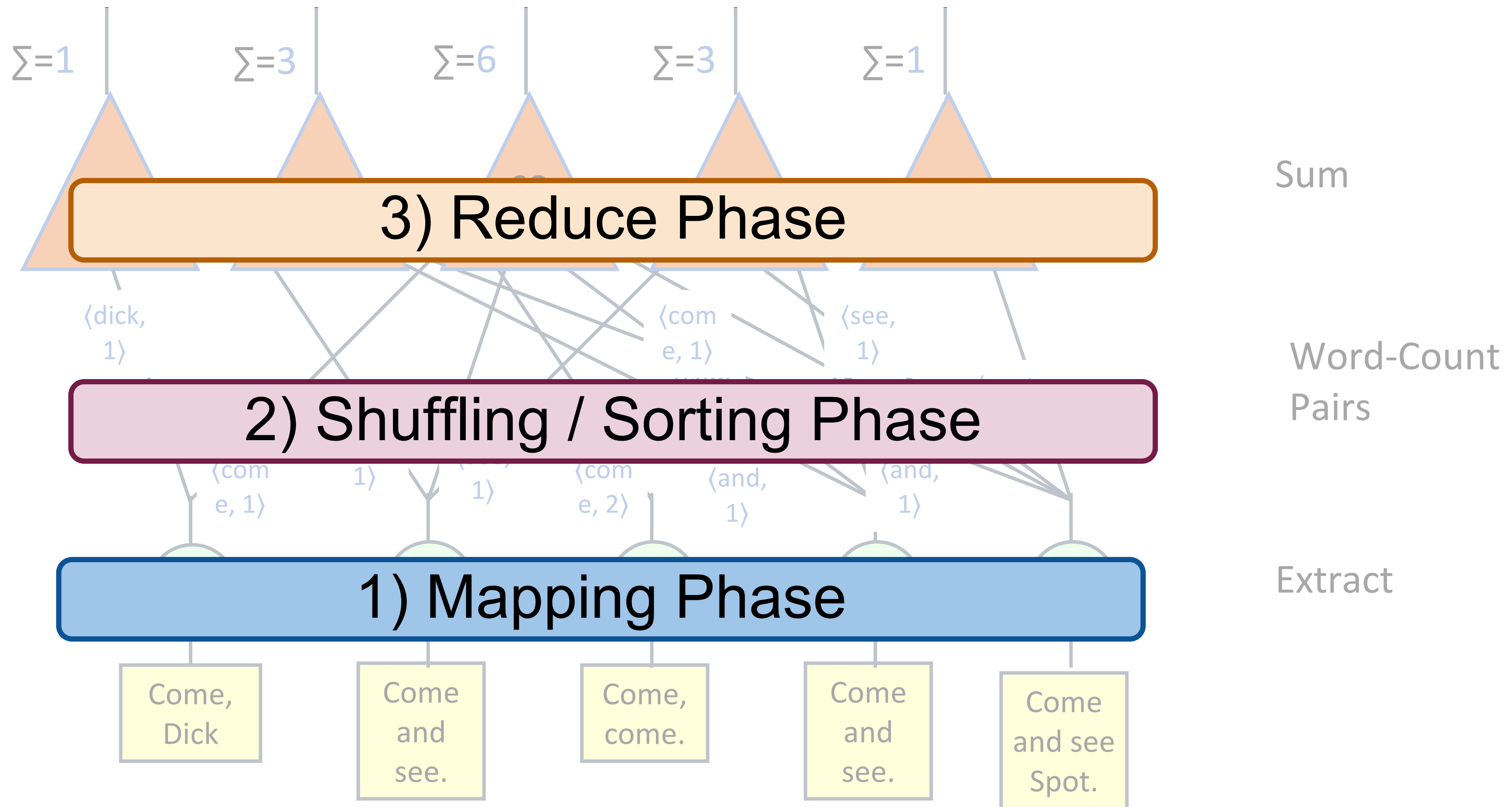
- Requirements
  - Programmer must supply Mapper & Reducer classes
- Mapper
  - Steps through file one line at a time
  - Code generates sequence of <key, value> pairs
  - Default types for keys & values are strings
    - Can use anything "writable", lots of conversion methods
- Shuffling/Sorting
  - MapReduce's built in aggregation by key
- Reducer
  - Given key + iterator that generates sequence of values
  - Generate one or more <key, value> pairs

Example I MapReduce



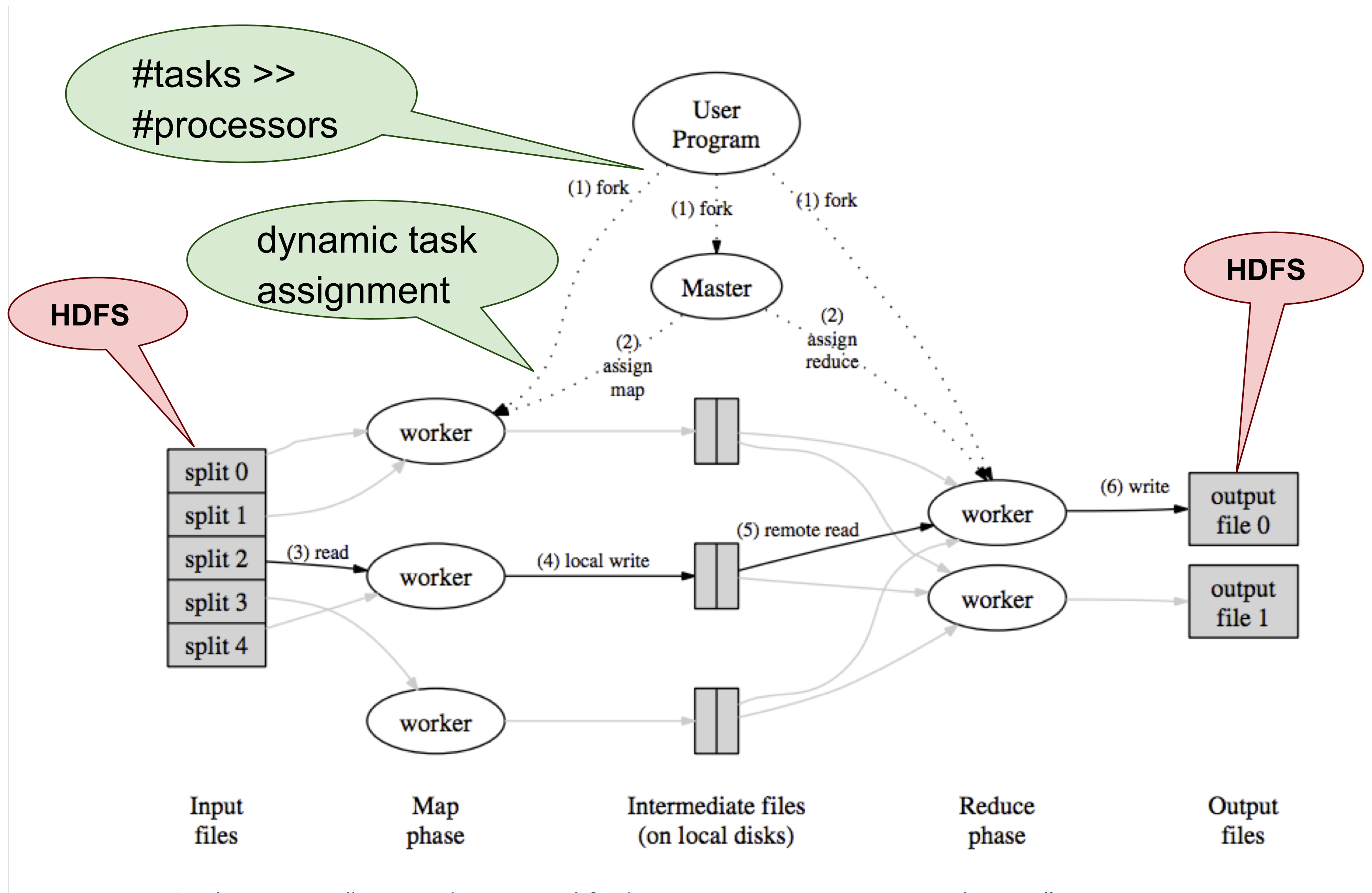
- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

Example I MapReduce



- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

MapReduce Execution



Dean & Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters", OSDI 2004

Distributed Systems

15-440/640

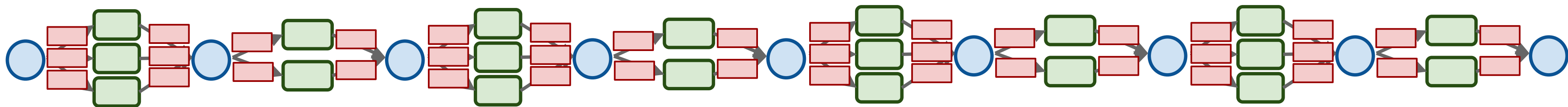
Fall 2018

17 – Fault-tolerant in-memory computation

Readings: “Resilient Distributed Datasets” Paper, Optional: “Immutability Changes Everything”

Limitations of MapReduce

Real-world applications require iterating MapReduce steps



Each iteration steps is small.

But: we need many iterations

⇒ 90% spent on I/O to disks and over network

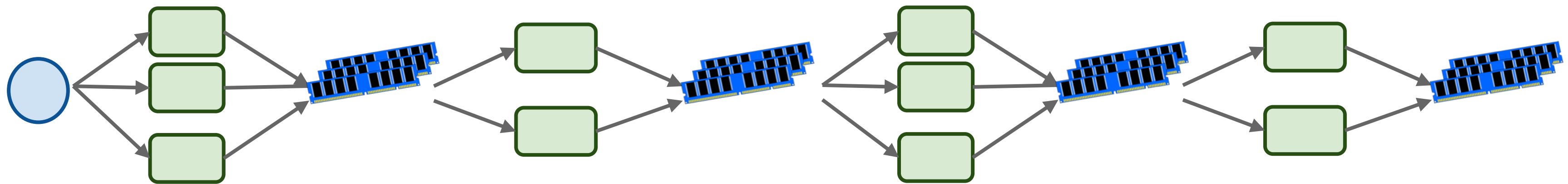
⇒ 10% spent computing actual results

Does not work for iterative applications
(⇒ distributed machine learning)

In-Memory Computation

Berkeley Extensions to Hadoop (⇒ Apache Spark)

Key idea: keep and share data sets in main memory



How to build **fault-tolerant** and **efficient** system?

Fault tolerance techniques from lectures so far?

Traditional fault-tolerance approaches

- Logging to persistent storage
- Replicating data across nodes (ideally: also to persistent storage)
- Checkpointing (checkpoints need to be stored persistently)

Spark Approach: RDDs and Lineage

Resilient **D**istributed **D**atasets

Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. NSDI 2012.

- Limit update interface to coarse-grained operations
- Efficient fault recovery using **lineage**
  - RDDs are immutable and partitioned across many nodes
  - Apply course-grained operations to every partition in parallel

Why immutability?

- Enables lineage
  - Recreate any RDD any time
  - More strictly: RDDs need to be deterministic functions of input
- Simplifies consistency
  - Caching and sharing RDDs across Spark nodes
- Compatibility with storage interface (HDFS)
  - HDFS chunks are append only

Spark Real World Challenges

RDD Lineage

- What if lineage grows really large?
  - manual checkpointing on HDFS

RDDs Immutability

- Deterministic functions of input
  - how to incorporate randomness?

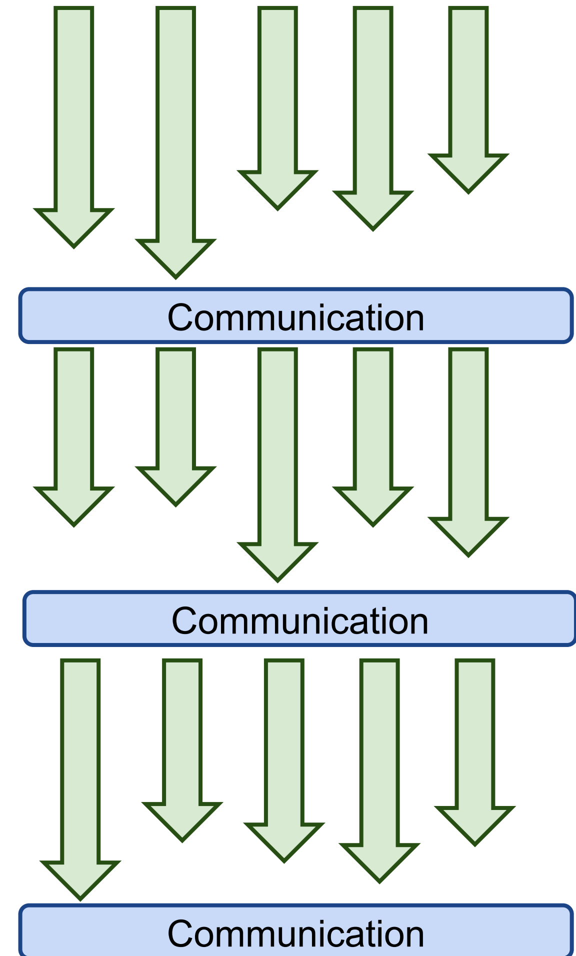
Other design implications?

- Needs lots of memory (might not be able to run your workload)
- High overhead: copying data (no mutate-in-place)

BSP computation abstraction

- Surprising power of iterations
  - (e.g., iterative Map/Reduce)
- Explained by theory of bulk synchronous parallel (BSP) model

<u>Theorem (Leslie Valiant,1990):</u>
"Any distributed system can be emulated as local work + message passing" (=BSP).



Spark implements BSP approximately

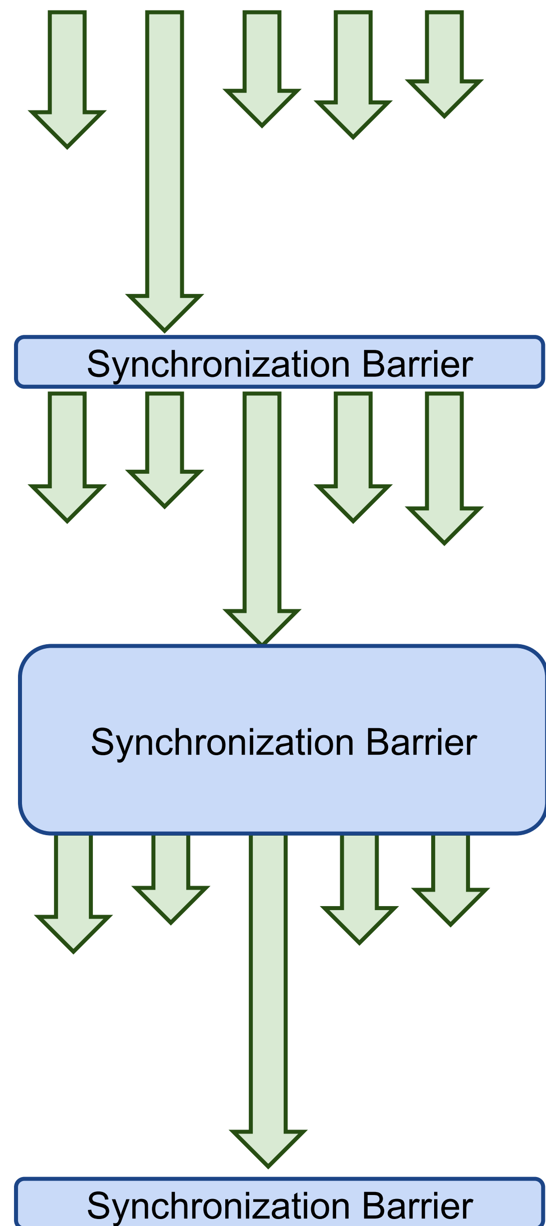
Challenge of Synchronization Overhead

BSP model:

- No computation during barrier
- No communication during computation

Fundamental limitation in BSP model

Constantly waiting for **stragglers**

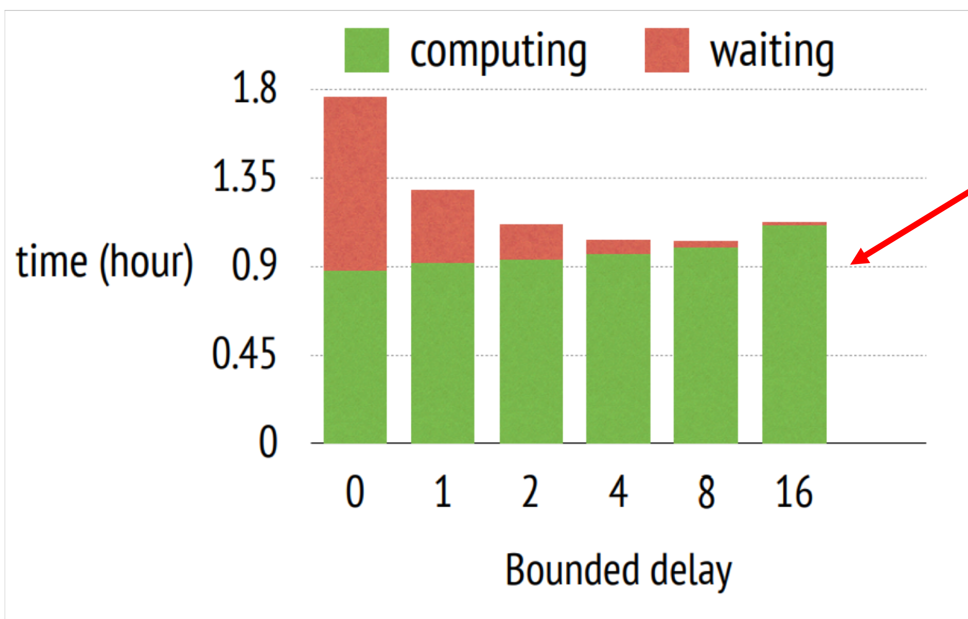


Do we need a new programming model?

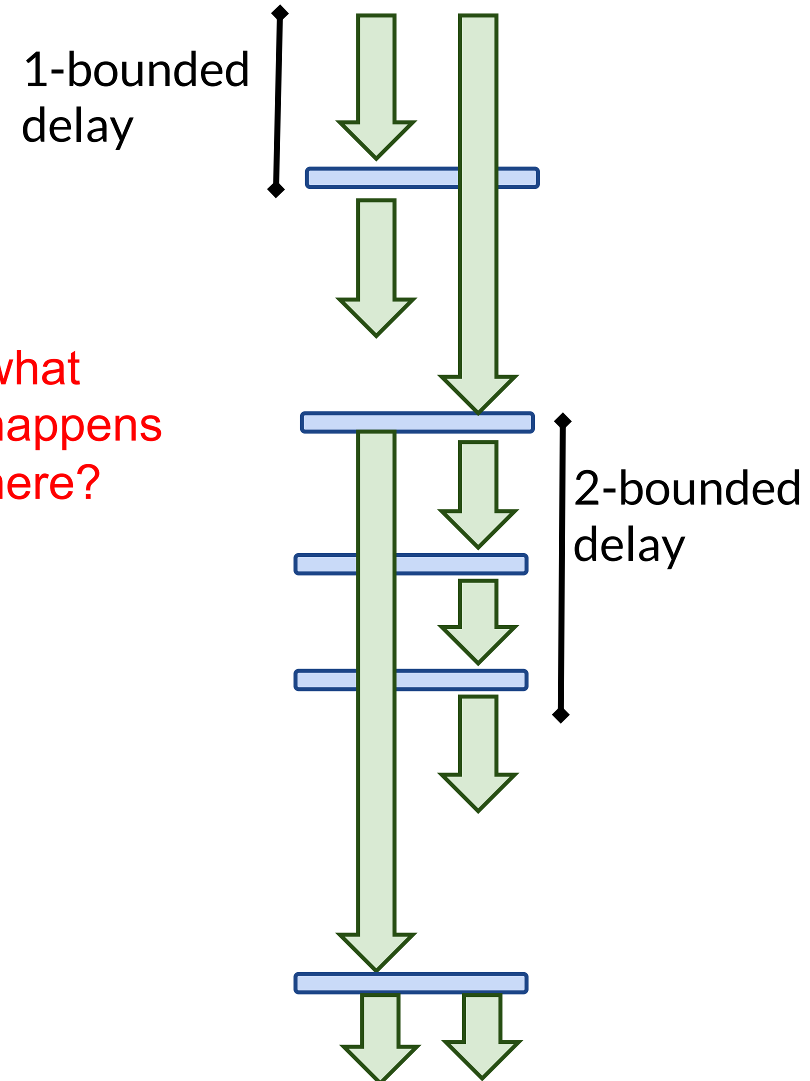
Bounded-delay BSP for Distributed ML

Bound stale state by N steps:

⇒ N-bounded delay BSP



what happens here?



From: Li et al, Scaling Distributed Machine Learning with the Parameter Server OSDI 2014

Distributed Systems

15-440/640

Fall 2018

18 – Internet Content Delivery Case Study: DNS & CDNs

Readings: Tanenbaum 5.1-5.5, 7.6. Optional readings: readings linked from website

1) How to map human-readable names (URLs) to server locations (IPs)?

The New York Times

World U.S. Politics N.Y. Business Opinion Tech Science Health Sports Olympics Arts Style Food Travel Magazine T Magazine

Sunday, February 11, 2018 | Today's Paper | Video | 54°F | Shanghai -0.05%

Abuse Claims Against Aide Further Roil White House

By PETER BAKER and MAGGIE HABERMAN 44 minutes ago

- While President Trump has expressed sympathy for an aide who resigned after spousal abuse allegations, some officials said the aide lied and deserved to be pushed out.
- And the status of the chief of staff, John F. Kelly, remains unclear as Mr. Trump's public sentiments conflict with what he is said to have expressed privately.

PYEONGCHANG 2018

Doug Mills/The New York Times

Kim Jong-un's Sister Upstages Mike Pence at the Olympics

Without a word, only flashing smiles, Kim Yo-jong outflanked Vice President Mike Pence in diplomacy.

EDITORIAL

Trouble in Maldives Paradise Could Become a Global Threat

A power struggle in the tiny nation could pull India and China into conflict.

Living Abroad Taught Me to Love America

By JANINE di GIOVANNI

Many liberals talk about moving to another country to avoid Trump. I did the opposite.

- Editorial: The Problem with Parole
- Leonhardt: Trumpism for Thee, but Not for Me

Opinion

The Biggest Valentine's Day Myths of All Time

By SARAH HUTTO

Cupid can in fact be thwarted.

Op-Docs: Have Head Shot, Seeking Wife

By ZUKI JUNO TOBGYE

Is a single photo enough to find your soul mate?

- Blow: G.O.P. Visions of Tectonic Realignment
- Democrats Can Win on Immigration
- America's Real Digital Divide
- Channeling Putin in Cairo
- For a Better Marriage, Act Like a Single Person
- Sharing the Plane Cabin With Animals

Internet "edge"

Internet "core"

Users

2) How to deliver content quickly & reliably?

Internet Name Discovery

Challenges/Goals:

- Scalability
- Decentralized maintenance
- Robustness
- Global scope

RR format: **(class, name, value, type, ttl)**

Basically, only one class: Internet (IN)

Types for IN class:

- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is name of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

Choosing the Time-To-Live

Common practices

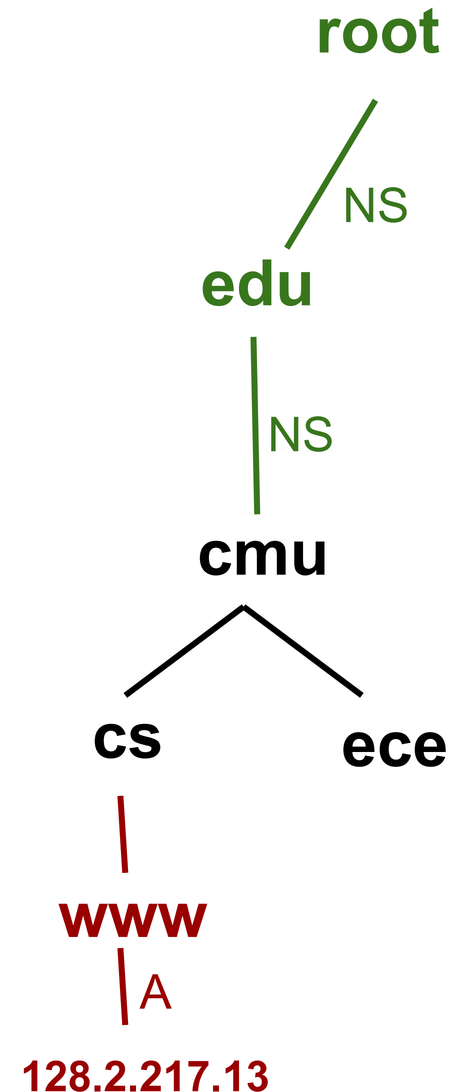
Top-level NS records: very high TTL
- alleviate load on root

Intermediary NS records: high TTL

A records: small TTL (<7200s)
- consistency concerns

Some A records: tiny TTL (<30s)
- fault tolerance, load balancing

Remember security implications when choosing TTLs!



Retrieving Web Content

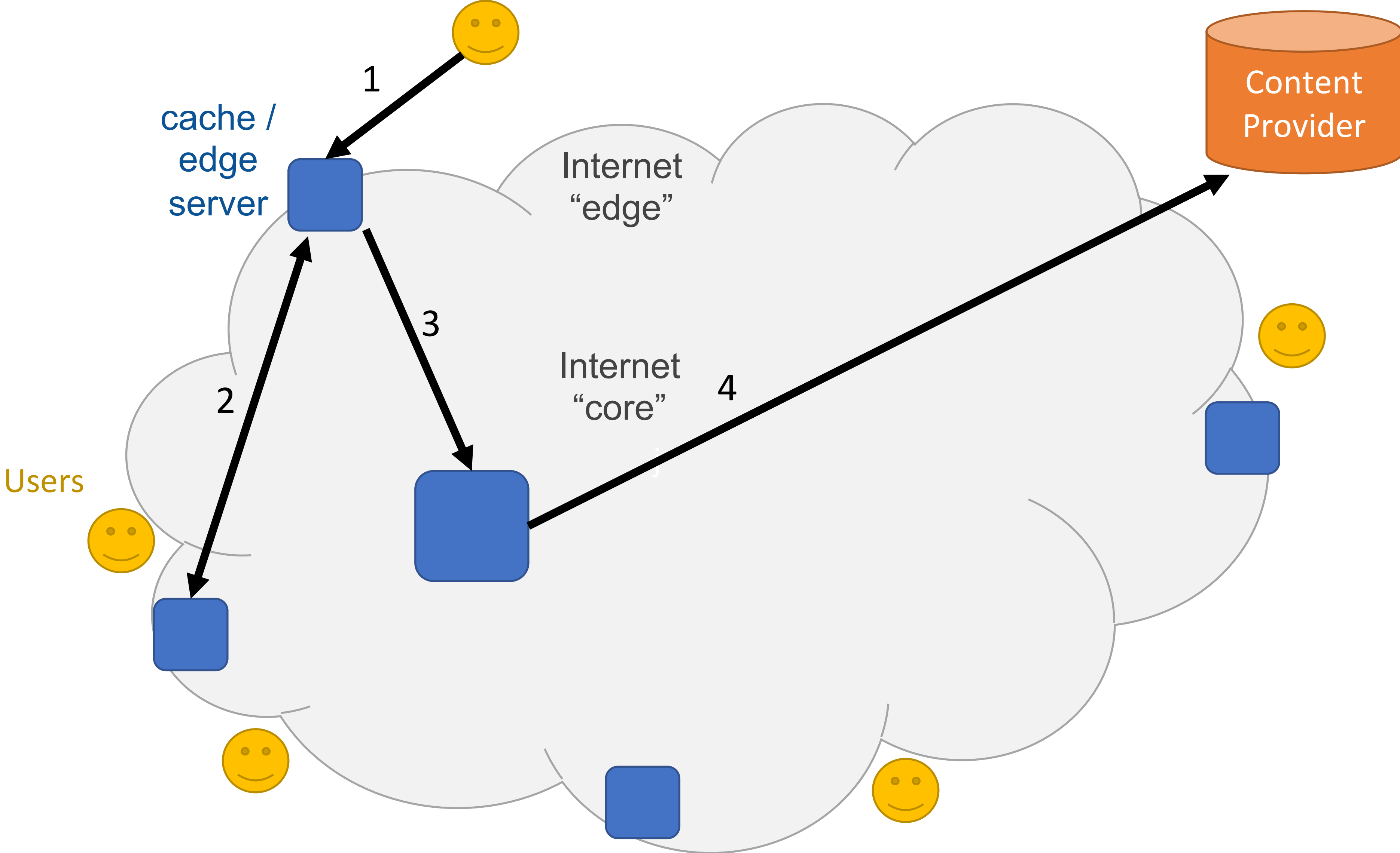
- Many (typically small) objects per page
- File sizes are heavy-tailed
- Embedded references

Lots of objects & TCP
- 3-way handshake
- Lots of slow starts
- Even worse: TLS

Why does this matter for performance?

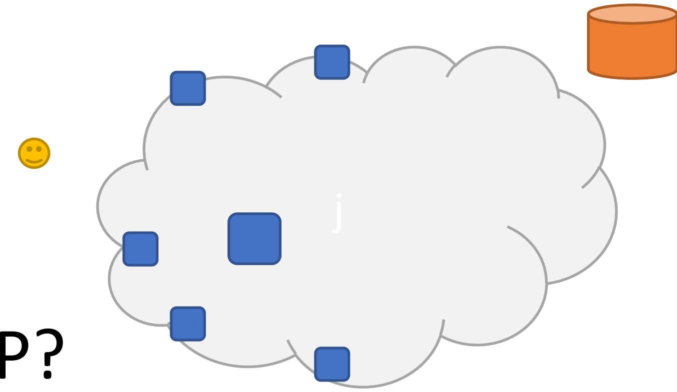
- Content Delivery Network (CDNs)
  - The world's largest distributed caching systems
  - Key for Internet performance
  - Explosive growth

A Typical CDN



Directing Users to CDNs

- Which PoP?
  - Best "performance" for this specific user
    - Based on Geography? RTT?
    - Throughput? Load?

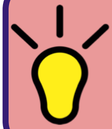


- How to direct user requests to the PoP?
  - As part of routing → anycast (= as part of IP routing)
  - As part of application → HTTP redirect
  - **As part of naming → DNS**

    (e.g., CNAME that is resolved via CDN's name server)

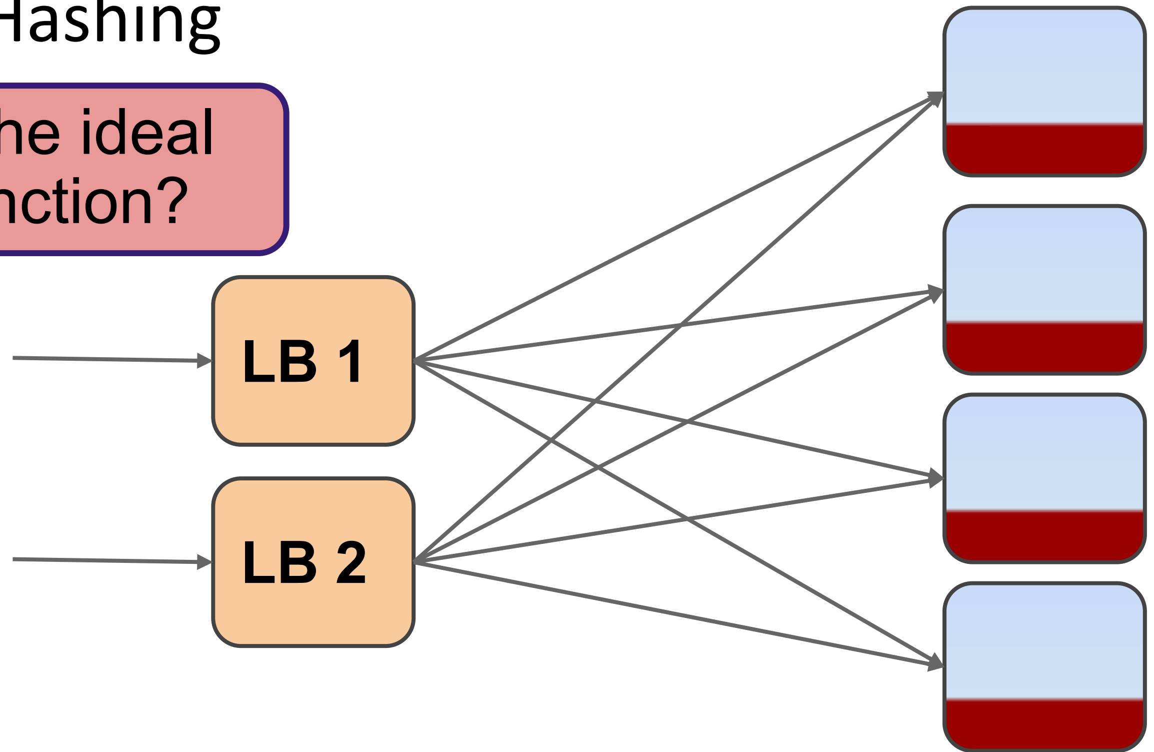
Actual CDN Load Balancer

Idea 4: Consistent Hashing



Properties of the ideal CDN hash function?

"View" = subset of all servers that are visible to LB



Desired properties

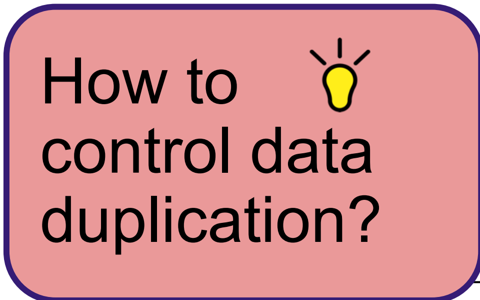
Load: over all views, # of objects / server is small (and ~uniform)

Spread: over all views, # of servers / obj is small (and ~uniform)

Smoothness: little impact when servers are added/removed

Consistent Hashing Example

Rule: A key is stored at its successor: node with next higher or equal ID



Cache Update Propagation Techniques
(a.k.a Cache Coherence Protocols)

1. Enforce Read-Only (Immutable Objects) — Spark
2. **Broadcast Invalidations** — CDNs
3. **Check on Use** — AFS1, HTTP1
4. Callbacks — AFS2
5. **TTLs (“Faith-based Caching”)** — DNS
6. Leases (generalize check on use and callbacks) — P3

Summary on CDNs

- Across wide-area Internet: **caching is the only way** to improve latency
- Caching aggressively used both by DNS and CDNs
- DNS resolvers → how does RR retrieval work?
- CDNs → how does content retrieval work?
- Consistent hashes and update propagation techniques

Distributed Systems

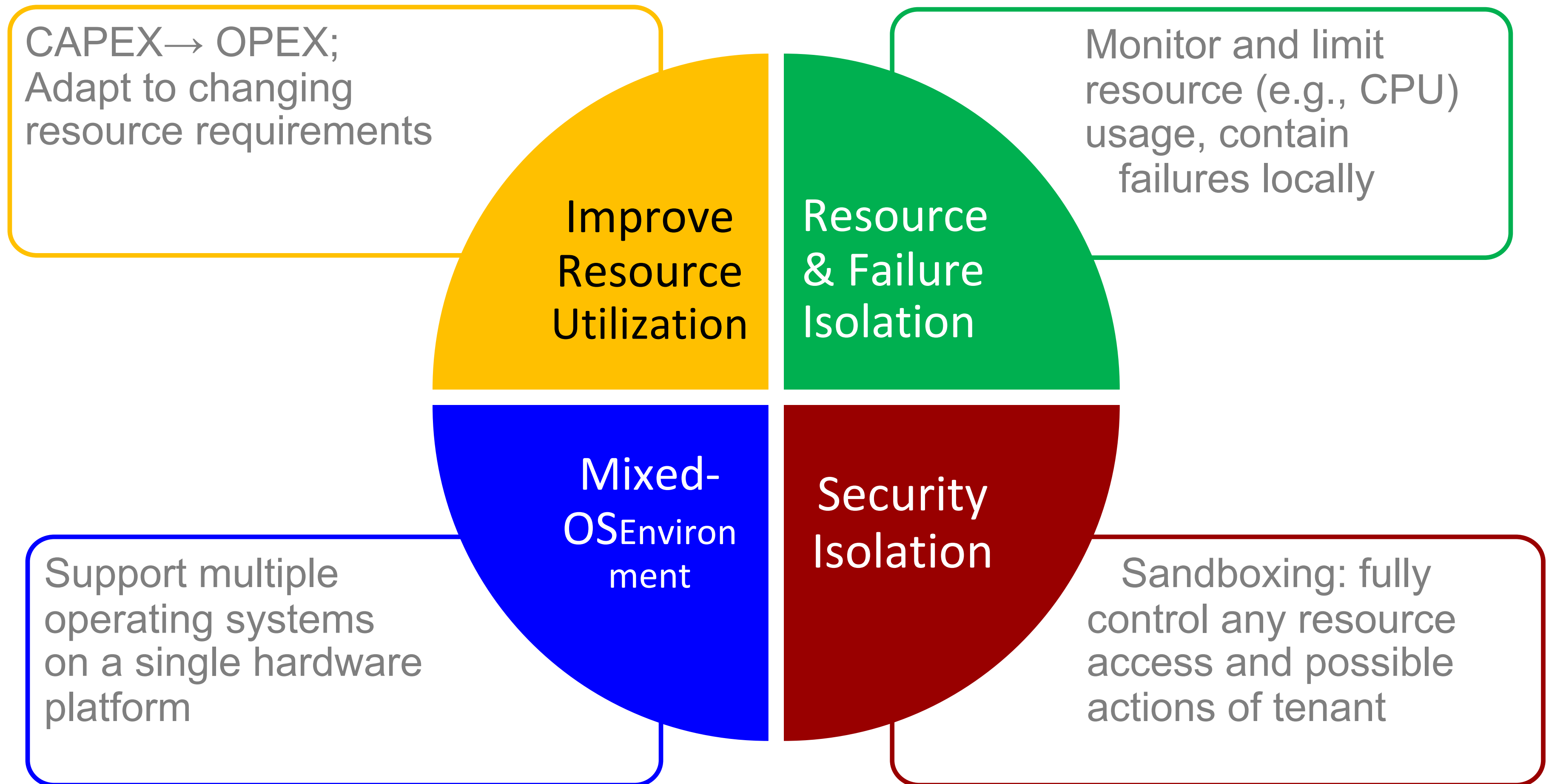
15-440/640

Fall 2018

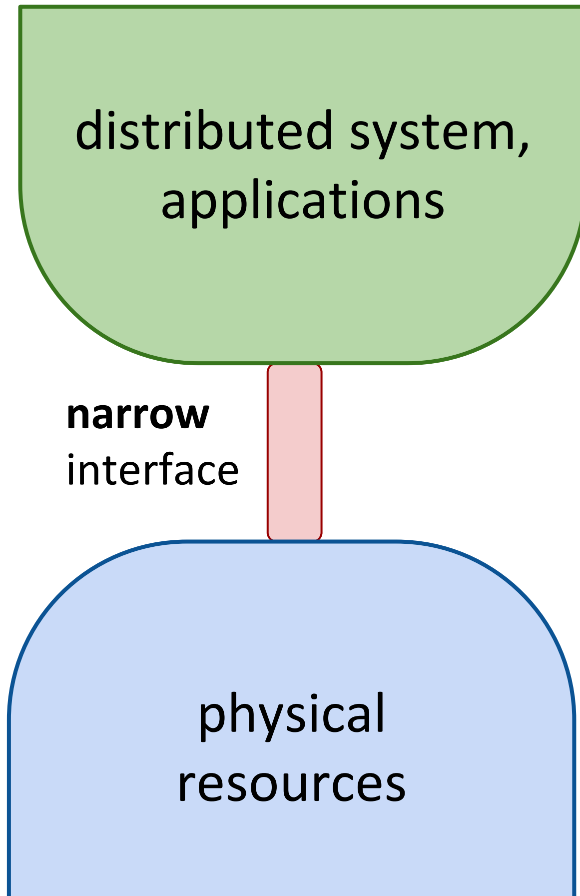
20– Virtualization Techniques

Readings: book chapter on Virtual Machines from the Wisconsin OS book

Reasons for Virtualization



Virtualization Techniques



Separate

- physical characteristics of resources
- from the way in which other systems, applications, or end users interact

Why Is Hardware Special?

Narrow & stable waistline critical

- narrow: freer innovation
- narrow: vendor neutrality
- stable: longevity / ubiquity

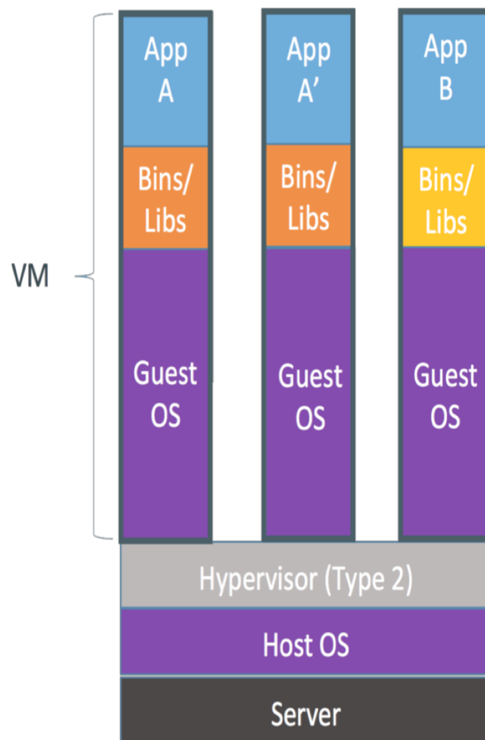
Wide interfaces → brittle abstractions

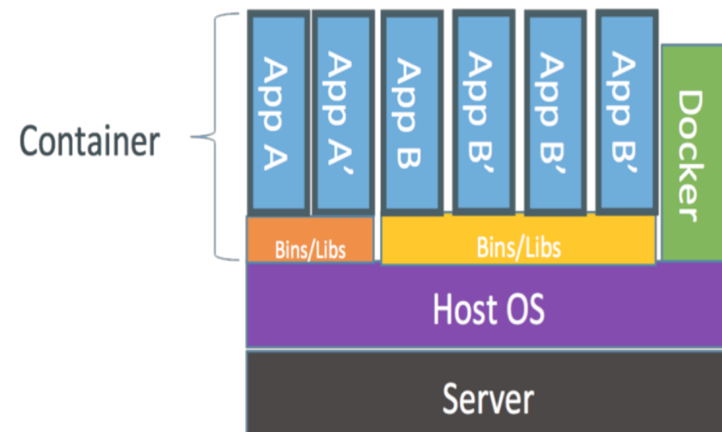
- hard to: deploy, sustain, scale
- e.g., software interface: processes

Adapted from: Ken Birman

Types of Virtualization

- System virtualization
  - Virtualizing the entire hardware interface
- Container virtualization
  - Virtualizing OS resources between processes





Requirements on VMs

- Isolation
  - Fault isolation
  - Performance isolation (+ software isolation, …)
- Encapsulation
  - Cleanly capture all VM state
  - Enables VM snapshots, clones
- Portability
  - Independent of physical hardware
  - Enables migration of live, running VMs (freeze, suspend,…)
  - Clone VMs easily, make copies
- Interposition
  - Transformations on **instructions, memory, I/O**
  - Enables transparent resource overcommitment, compression, replication …

Resource & Failure Isolation

Mixed-OS Environment

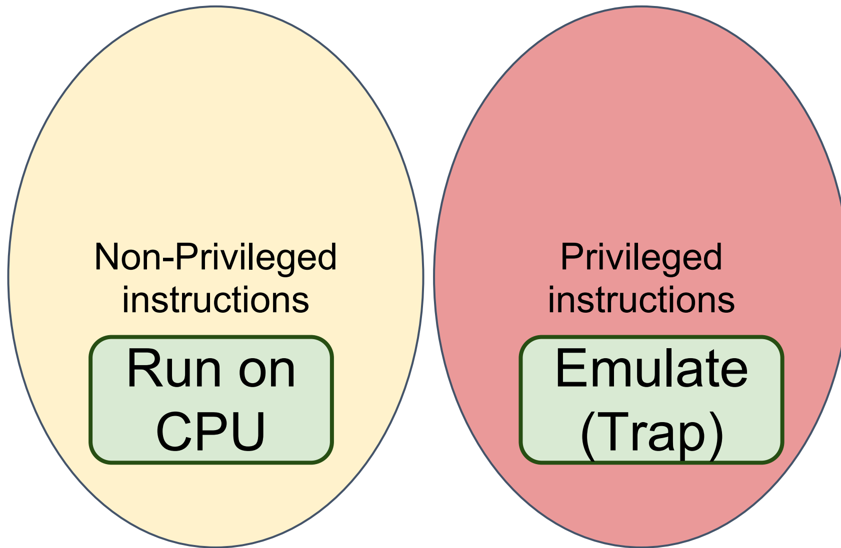
Improved Resource Utilization

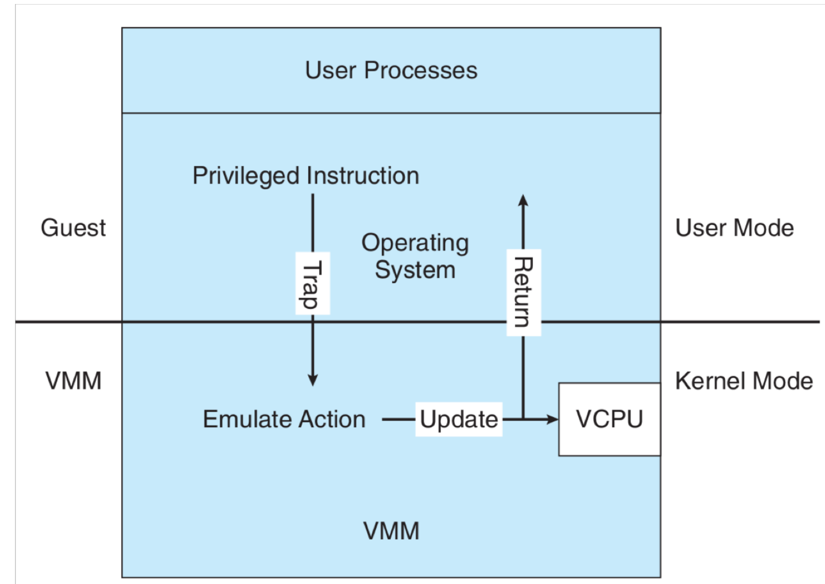
Security Isolation

How to implement interposition for CPU, memory, I/O?

Adapted from: Eyal DeLara

Efficient CPU Virtualization





- **Non-privileged instructions** (e.g., Load from mem): Run as native machine
- **Privileged instructions** (e.g., Update CPU state, Manipulate page table): Trap to VMM

This is called Trap and Emulate
→ Full Control for VMM

More complex in reality (some privileged instructions don’t trap) → Processor support VT-x, AMD-V

Why Container Virtualization?

Overhead associated with deploying on VMs

- I/O overhead
- OS-startup overhead per VM
- Memory/Disk overhead (duplicate data)

Overhead becomes dominant at scale: thousands of VMs / server

Perception: VM have too much overhead!

New idea:

- Multiple isolated instances of programs
- Running in user-space (shared kernel)
- Instances see only resources (files, devices) assigned to their container

Other names: OS-level virtualization, partitions, jails (FreeBSD jail, chroot jail)

Requirements on Containers

- Isolation and encapsulation
  - Fault and performance isolation
  - Encapsulation of environment, libraries, etc.

Resource & Failure Isolation

- Low overhead
  - Fast instantiation / startup
  - Small per-operation overhead (I/O, ..)

Improved Resource Utilization

- Reduced Portability

Mixed-OS Environment

- ~~Interposition~~ (no hypervisor)

Security Isolation

Implementation

Key problems:

- Isolating which resources containers see
  → Linux namespace
- Isolating resource usage
  → Linux control groups
- Efficient per-container filesystems
  → Linux OverlayFS

Summary

VMs

Strengths: strong isolation guarantees, can run different OSs

VM migration practical

Weaknesses: OS startup, disk,memory, and hypervisor overhead

Containers

Strength: fast startup times, negligible I/O overheads, very high density

Weaknesses: weak security isolation

In practice: techniques complement each other

Use VMs to isolate between different users, and containers to isolate different applications/services of a single user

Distributed Systems

15-440/640

Fall 2018

21 – Byzantine Fault Tolerance

Readings: Tanenbaum pages 449 - 460. PBFT paper.

Agreement in Faulty Systems

Possible characteristics of the underlying system:

1. Synchronous versus asynchronous systems.
   - A system is synchronized if the process operation in lock-step mode. Otherwise, it is asynchronous.
2. Communication delay is bounded or not.
3. Message delivery is ordered or not.
4. Message transmission is done through unicasting or multicasting.

Agreement in Faulty Systems

Message ordering

Process behavior	Communication delay	Unordered: Unicast	Unordered: Multicast	Ordered: Unicast	Ordered: Multicast
Synchronous	Bounded	X	X	X	X
Synchronous	Unbounded			X	X
Asynchronous	Bounded				X
Asynchronous	Unbounded				X

Message transmission

Circumstances under which distributed agreement can be reached. Note that most distributed systems assume that

1. processes behave asynchronously
2. messages are unicast
3. communication delays are unbounded (see red blocks)

What do Byzantine Failures Look Like?

Many things can go wrong…

Communication

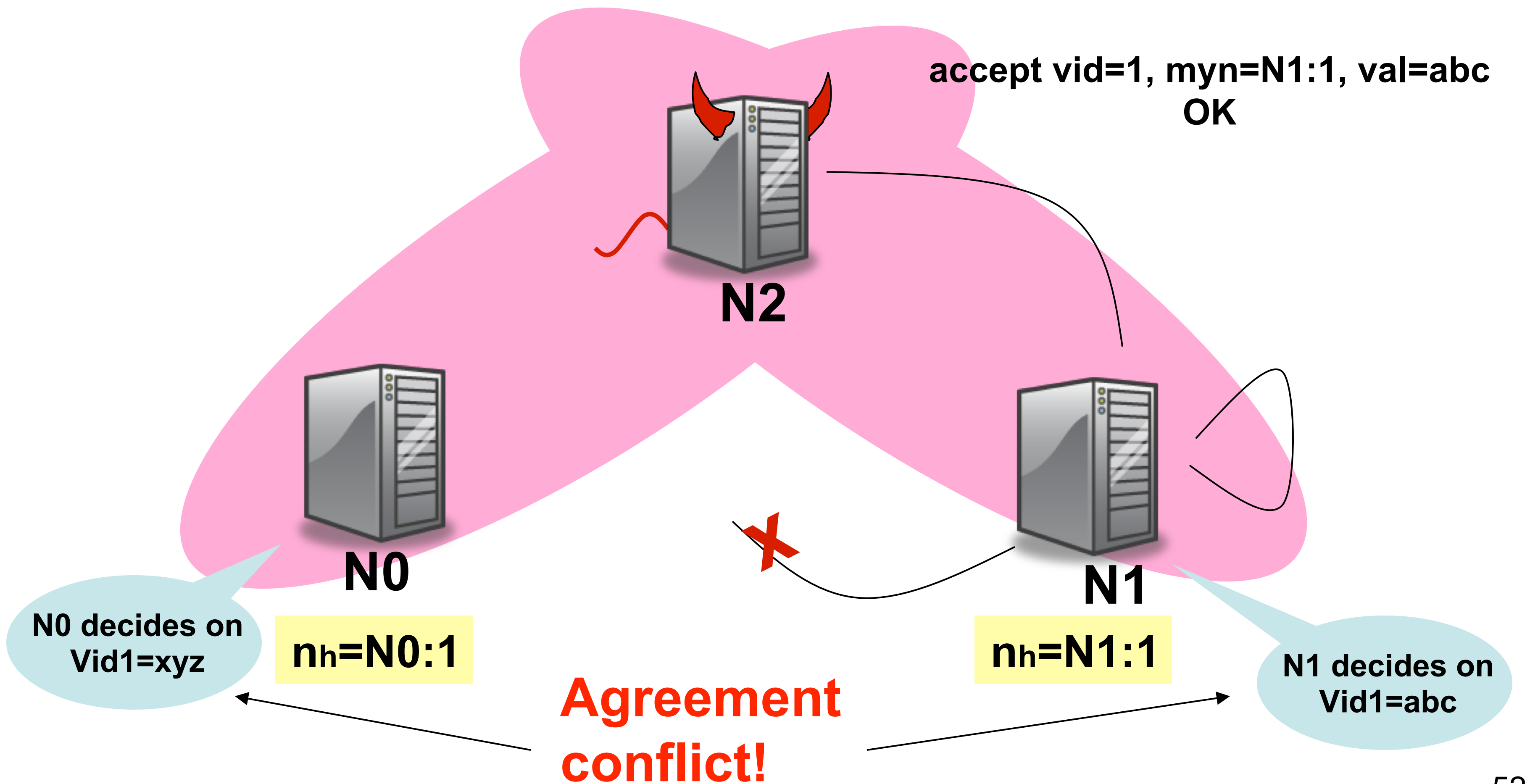
- Messages lost or delayed for arbitrary time
- Adversary can intercept messages

Processes

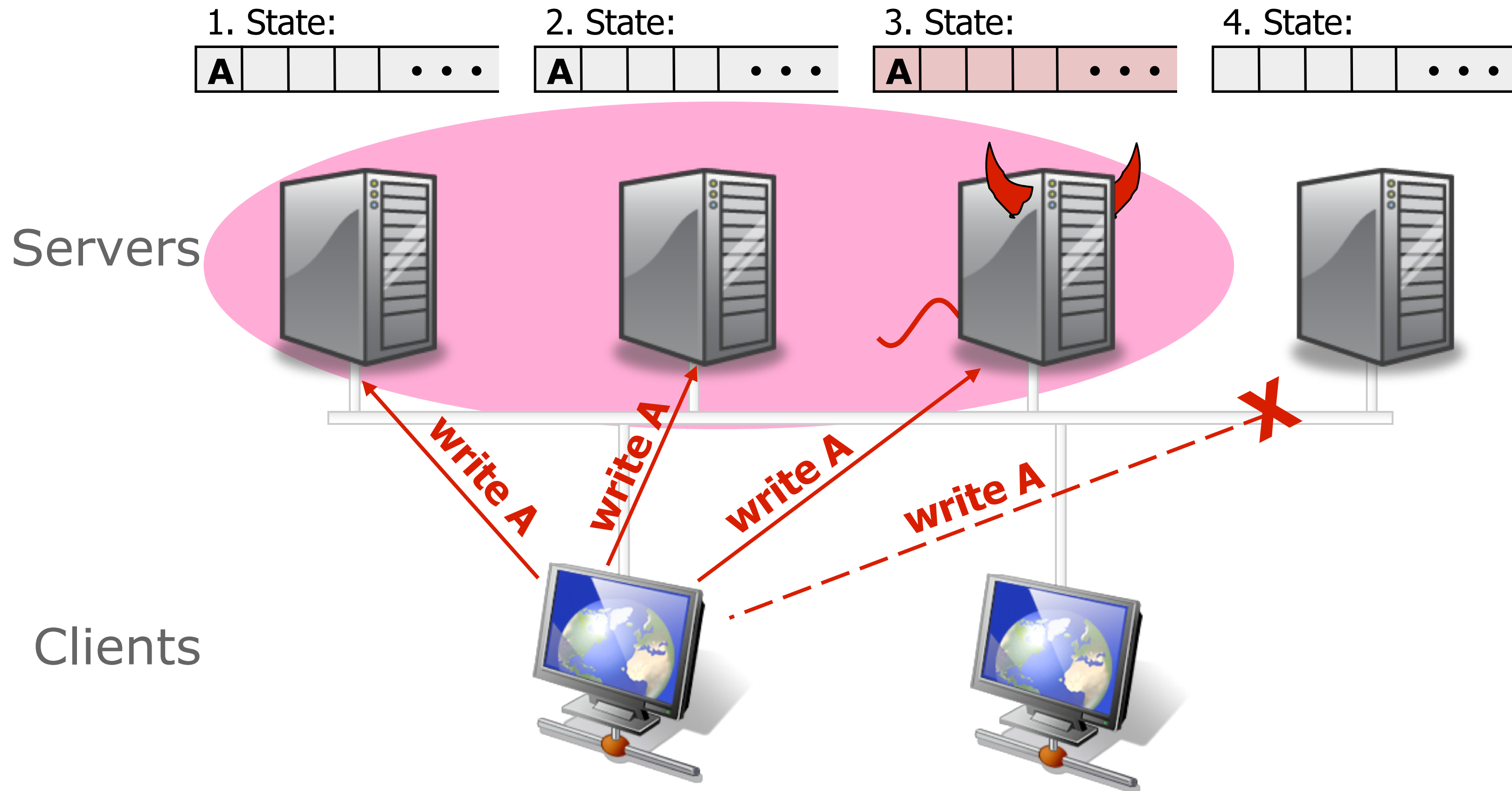
- Can fail or team up to produce wrong results

Agreement very hard, when possible to achieve?

Paxos under Byzantine faults

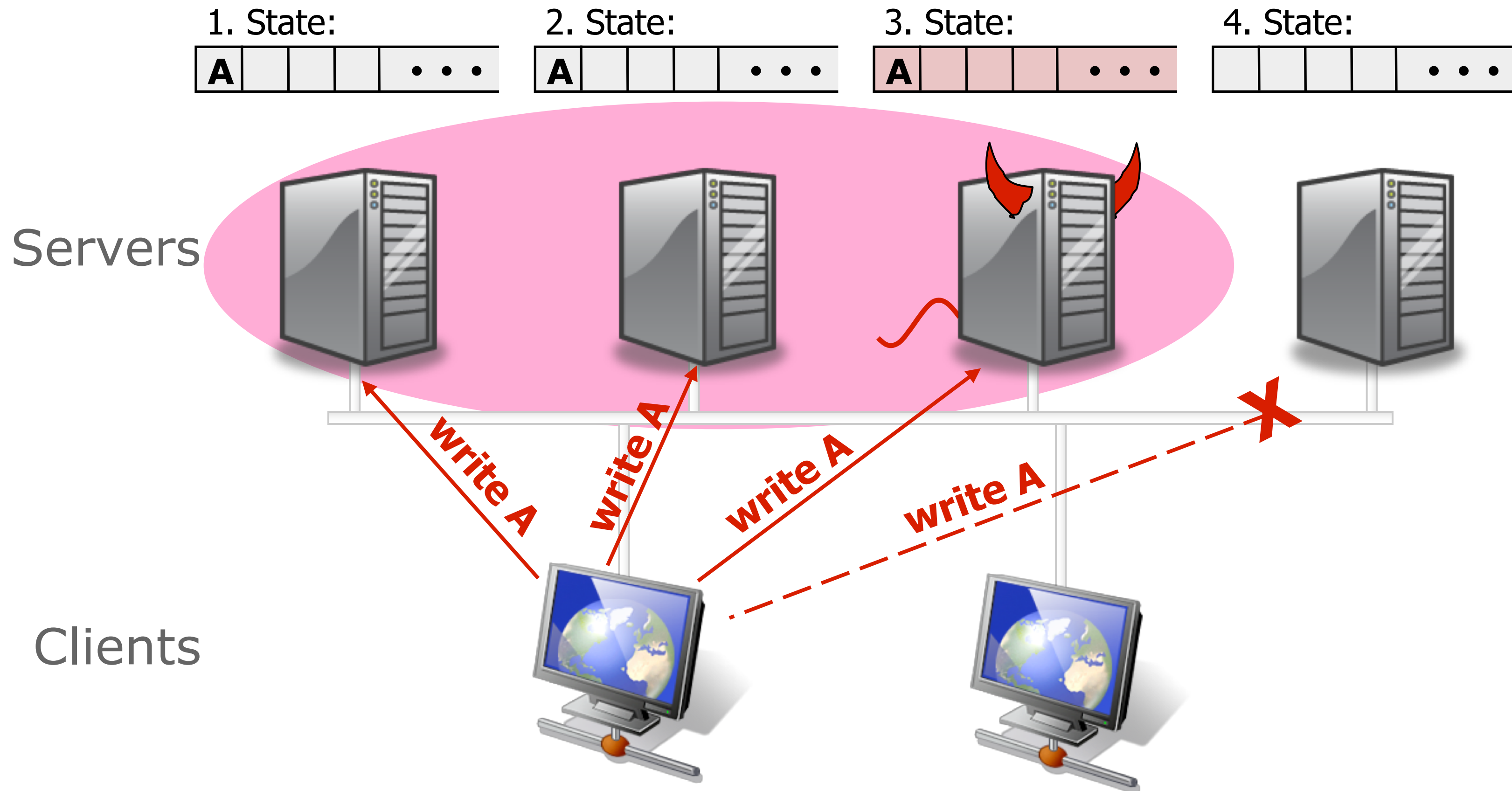


BFT: What Quorum Size Do We Need?



For liveness, the quorum size must be at most N - f

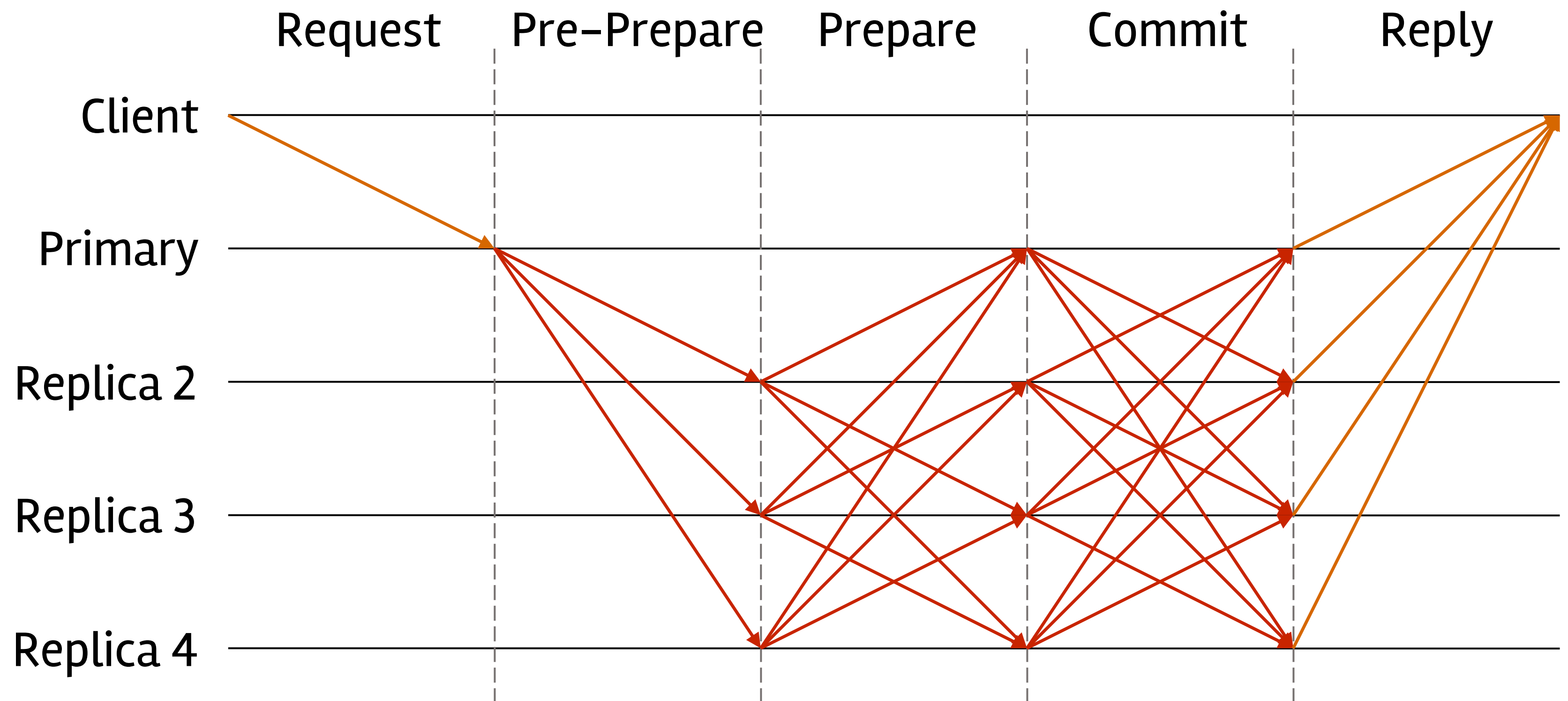
BFT: What Quorum Size Do We Need?



For correctness, any two quorums must intersect at least one honest node: (N-f) + (N-f) - N >= f+1 N >= 3f+1

Practical Byzantine Fault Tolerance

Quorum-based Byzantine consensus protocol



Normal Case

- Client waits for f+1 matching replies

Why f+1? What does this ensure?

- Ensures that at least one honest node has committed and executed

What does commit of at least one honest node ensure?

- Ensure 2f+1 matching commits

  $\Rightarrow$ At least f+1 honest nodes have committed

Distributed Systems

15-440 / 15-640
Fall 2018-Review

Blockchains

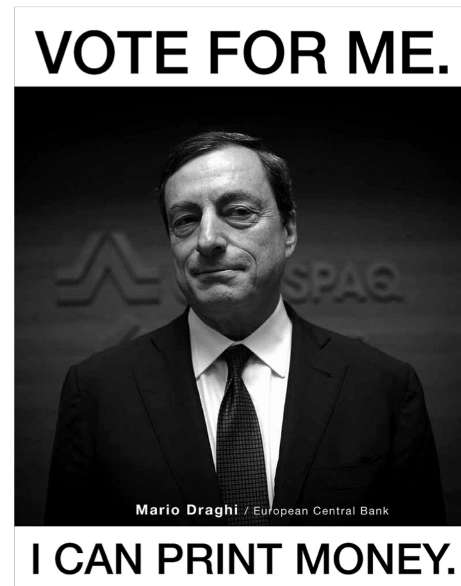
Motivation: Decentralized Transactions

- Traditional transactions
  - Via trusted entities like banks/mints/lawyers
  - Transactions sometimes need to be reversed
    - Disputes, stolen credit cards
    - Requires mediation and additional trust (merchant → customer)
  - Significant transaction costs
    - Prevent emerging use cases, e.g., micro payments

- Decentralized transactions
  - Are they possible? Can you think of some challenges?

Challenges of Decentralized Transactions

- Key problem: double spending
  - Daniel has $5 - buys a $5-drink and a $5-sandwich at the same time
- Someone needs to keep track of **ALL** transactions
  - Traditional currency: mints
  - Internet: P2P distributed data-structure



Solution: Secure, Distributed Ledgers

- Ledger:

every transaction ever made

- All participants need a copy

Steps to Maintain Distributed Ledger:

- 1) broadcast new transactions
- 2) each member collects transactions into a block
- 3) once block is full, broadcast, and move on

What if there's a block collision?

Adding Consensus to Distributed Ledgers

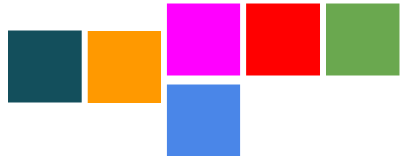
Steps to Maintain Distributed Ledger:

- 1) broadcast new transactions
- 2) each member collects transactions into a block
- 3) reach consensus on next block
- 4) continue with 1)

Sybil Attack: What if someone has many IP addresses?

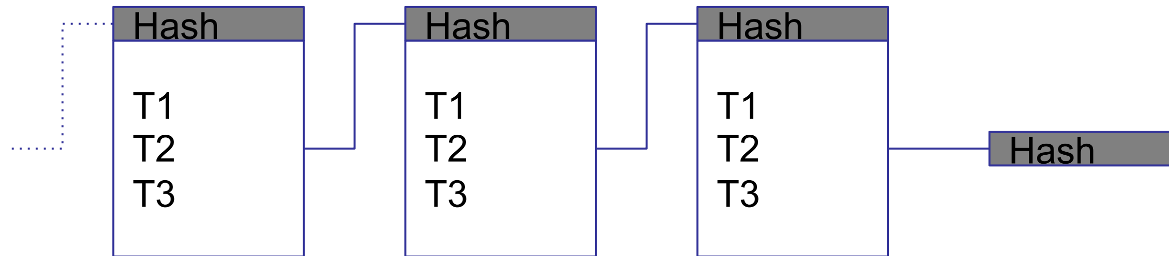
Solution: Blockchain Consensus

- 1) broadcast new transactions
- 2) each member collects transactions into a block
- 3) each member seeks proof-of-work for its block
  - proof-of-work (PoW): solve a computationally hard problem
- 4) member who finds PoW broadcasts block+PoW
- 5) other member check block, seek next PoW
- 6) consensus **over time**



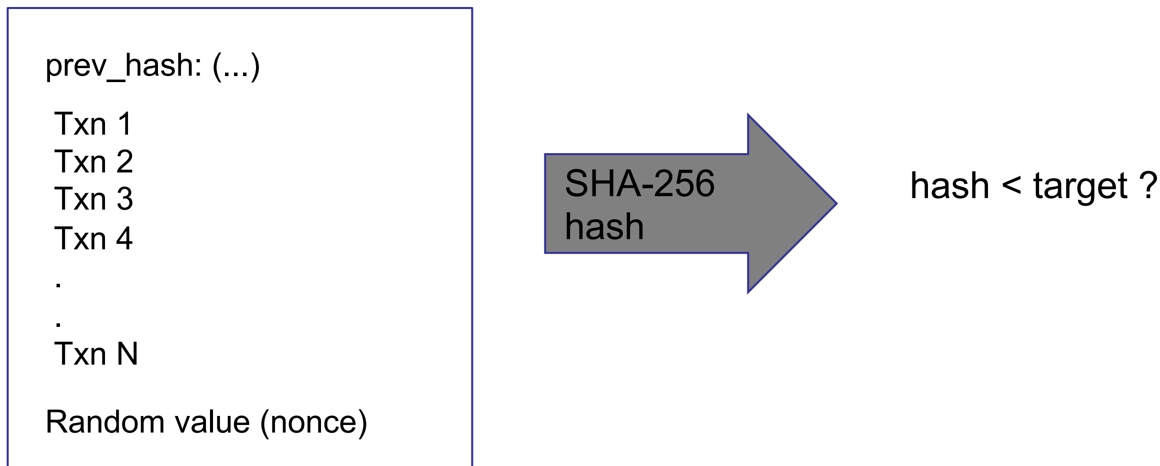
Blockchain in More Detail

- Blocks contain transactions
- Chain of blocks secured using cryptographic hashes
- Each block contains cryptographic hash of previous block
- Tampered block can easily be checked for



Blockchain Proof-of-Work

- Idea: one vote per CPU
- Hashcash cryptographic puzzle used in Bitcoin
- Find nonce such that SHA-256 hash of (block + nonce) has K leading zeros



Security Guarantee of a Blockchain

- To modify old transactions, proof of work has to be redone for all successive committed blocks
- 51 attack
  - If an organization has more than 51% of the total compute, it can choose which transactions get committed
  - Very hard to change older blocks even with a majority of computational power

Incentivizing proof of work (mining)

- Mining is the process of generating proof of work
- Miner adds reward to self at the beginning of the block
- If the miner's block gets added to the blockchain, miner receives a reward

Example: bitcoin

Ledger: transactions of bitcoin currency payments

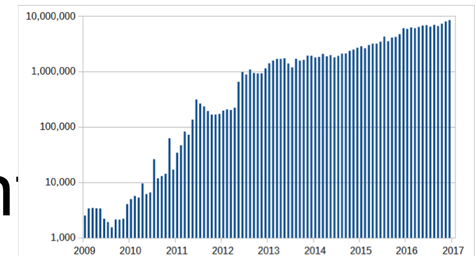
Reward: bitcoins

Introduced in 2009 by “Satoshi Nakamoto”

(not kown publicly)

In use today

(10 million transactions / mon



Example: Namecoin

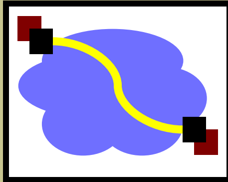
Ledger: Names and IP addresses of various servers, along with namecoin transactions

Reward: Namecoins, which are just like bitcoins

Introduced in 2011

Censor-free fully p2p naming system

“decentralized DNS”

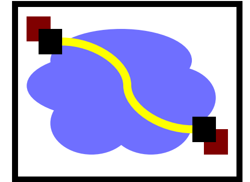


15-440 Distributed Systems

25 – Final Review (Part 2)
Security Protocols

Tuesday, Dec 4th, 2018

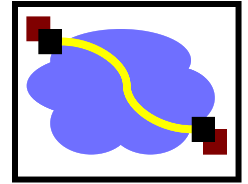
Logistical Updates



- HW4 - Due 12/4 (Tuesday) **NO LATE DAYS**

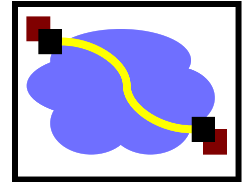
- Midterm II – Thursday 12/6, 10:30am – 11:50am
  - In CUC McConomy. Please come 10mins early.
  - We will be able to set up and will start on time!

What do we need for a secure communication channel?



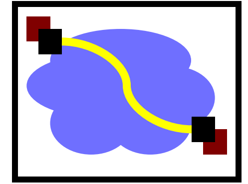
- Authentication (Who am I talking to?)
- Confidentiality (Is my data hidden?)
- Integrity (Has my data been modified?)
- Availability (Can I reach the destination?)

Example: Web access



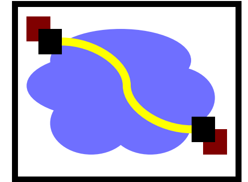
- Alice wants to connect to her bank to transfer some money...
- Alice wants to know ...
  - that she's really connected to her bank. Authentication
  - That nobody can observe her financial data Confidentiality
  - That nobody can modify her request Integrity
  - That nobody can steal her money! (A mix)
- The bank wants to know ...
  - That Alice is really Alice (or is authorized by Alice)
  - The same privacy things that Alice wants so they don't get sued or fined by the government.

How do we create secure channels?



- What tools do we have at hand?
- Hashing
  - e.g., SHA-1
- Secret-key cryptography, aka symmetric key.
  - e.g., AES
- Public-key cryptography
  - e.g., RSA

Secret Key Cryptography



- Given a key k and a message m
  - Two functions: Encryption (E), decryption (D)
  - ciphertext c = E(k, m)
  - plaintext m = D(k, c)
  - Both use the same key k.

“secure” channel

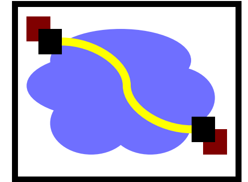
Alice
knows K

Bob.com
knows K

But... how does that help with authentication?

They both have to know a pre-shared key K before they start!

Symmetric Key: Confidentiality



- One-time Pad (OTP) is secure but usually impractical
  - Key is as long at the message
  - Keys cannot be reused (why?)

In practice, two types of ciphers are used that require only constant key length:

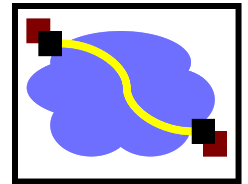
Stream Ciphers:

Ex: RC4, A5

Block Ciphers:

Ex: DES, AES, Blowfish

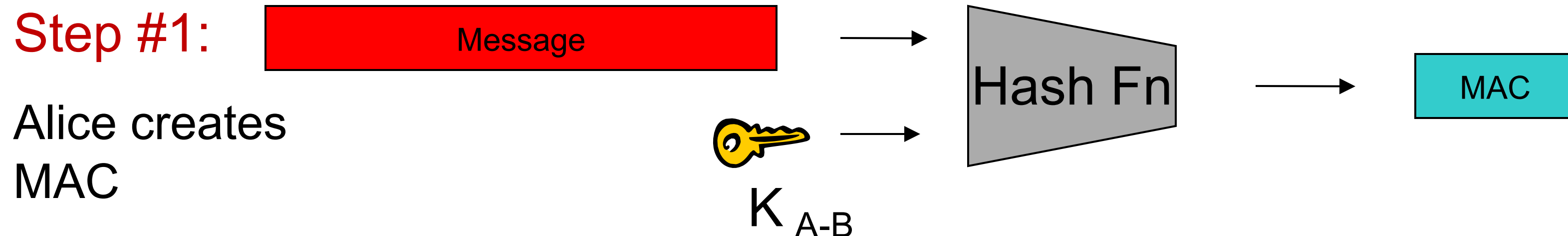
Symmetric Key: Integrity



- Hash Message Authentication Code (HMAC)

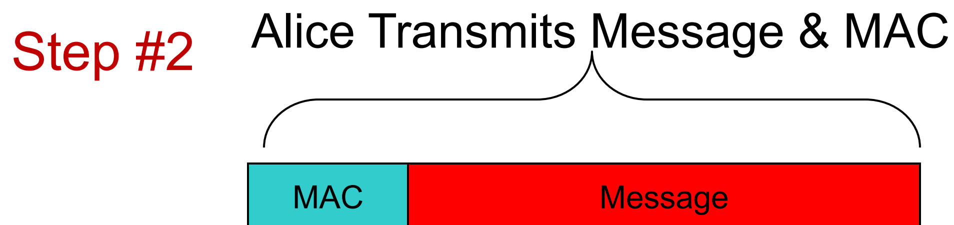
Step #1:

Alice creates MAC



Step #2

Alice Transmits Message & MAC

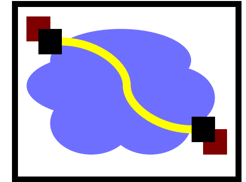


Step #3

Bob computes MAC with message and K_{A-B} to verify.

Why is this secure from a message integrity perspective?
How do properties of a hash function help us?

Symmetric Key: Authentication



- A "Nonce"
  - A random bitstring used only once. Alice sends nonce to Bob as a "challenge". Bob Replies with "fresh" MAC result.

Alice

Nonce

Bob

Nonce

Hash

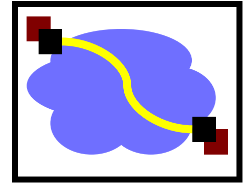
B4FE64

K_{A-B}

B4FE64

Performs same hash with K_{A-B} and compares results

Symmetric Key Crypto Review



- Confidentiality: Stream & Block Ciphers
- Integrity: HMAC
- Authentication: HMAC and Nonce

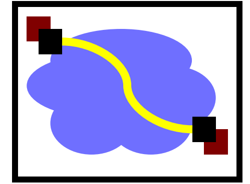
Questions??

Are we done? Not Really:

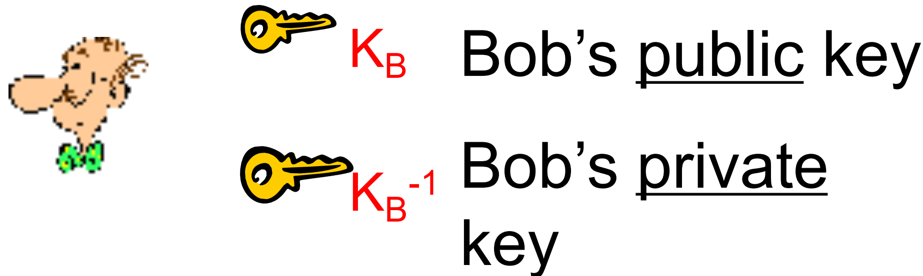
1) Number of keys scales as O(n^2)

2) How to securely share keys in the first place?

Asymmetric Key Crypto:

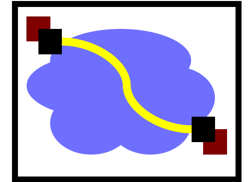


- Instead of shared keys, each person has a "key pair"



- The keys are inverses, so: $K_B^{-1}(K_B(m)) = m$

Asymmetric/Public Key Crypto:



- Given a key *k* and a message *m*
  - Two functions: Encryption (E), decryption (D)
  - ciphertext $c = E(K_B, m)$
  - plaintext $m = D(K_B^{-1}, c)$
  - Encryption and decryption use *different* keys!

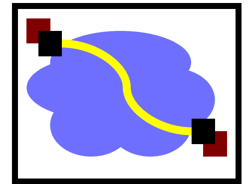
“secure” channel

Alice
Knows K_B

Bob.com
Knows K_B, K_B^{-1}

But how does Alice know that K_B means “Bob”?

Asymmetric Key: Confidentiality



K_B Bob's <u>public</u> key

K_B^{-1} Bob's <u>private</u> key

encryption algorithm

ciphertext

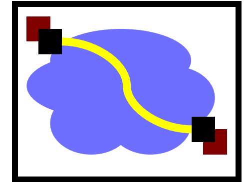
K_B (m)

decryption algorithm

plaintext message

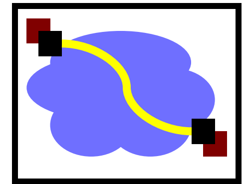
$m = K_B^{-1}(K_B(m))$

Asymmetric Key: Sign & Verify



- If we are given a message M, and a value S such that $K_B(S) = M$, what can we conclude?
  - The message must be from Bob, because it must be the case that $S = K_B^{-1}(M)$, and only Bob has K_B^{-1} !
- This gives us two primitives:
  - Sign (M) = $K_B^{-1}(M)$ = Signature S
  - Verify (S, M) = test($K_B(S) == M$)

Asymmetric Key: Integrity & Authentication

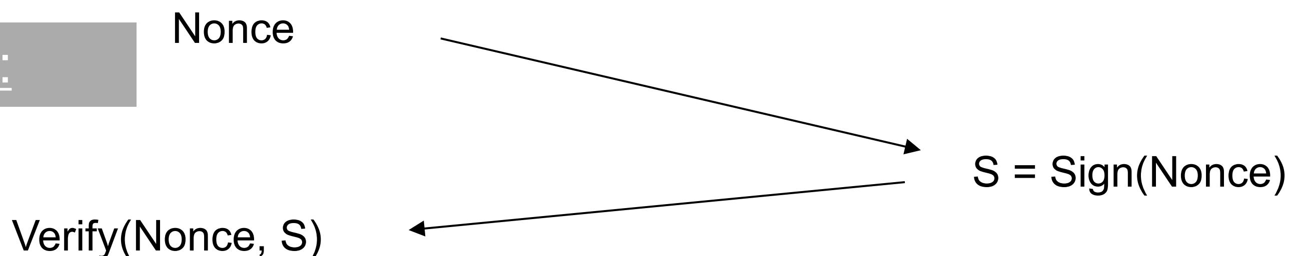


- We can use Sign() and Verify() in a similar manner as our HMAC in symmetric schemes.

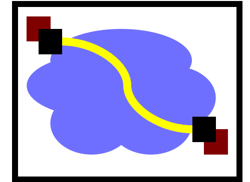
Integrity:

Receiver must only check Verify(M, S)

Authentication:



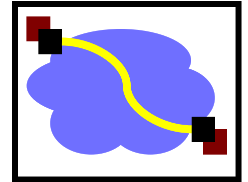
Asymmetric Key Review:



- Confidentiality: Encrypt with Public Key of Receiver
- Integrity: Sign message with private key of the sender
- Authentication: Entity being authenticated signs a nonce with private key, signature is then verified with the public key

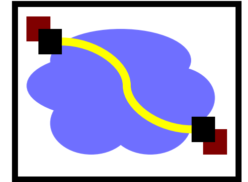
But, these operations are computationally expensive*

The Great Divide



	Symmetric Crypto: (Private key) Example: AES	Asymmetric Crypto: (Public key) Example: RSA
Requires a pre-shared secret between communicating parties?	Yes	No
Overall speed of cryptographic operations	Fast	Slow

One last “little detail”…



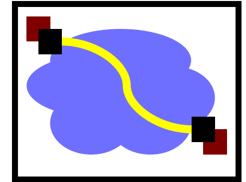
How do I get these keys in the first place??

Remember:

- Symmetric key primitives assumed Alice and Bob had already shared a key.
- Asymmetric key primitives assumed Alice knew Bob’s public key.

This may work with friends, but when was the last time you saw Amazon.com walking down the street?

Key Distribution Center (KDC)



Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?

$K_{A\text{-}KDC}$(A,B)

KDC generates R1

$K_{A\text{-}KDC}$(R1, $K_{B\text{-}KDC}$(A,R1))

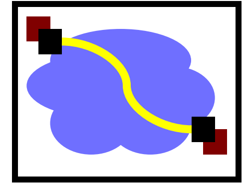
Alice knows R1

$K_{B\text{-}KDC}$(A,R1)

Bob knows to use R1 to communicate with Alice

Alice and Bob communicate: using R1 as *session key* for shared symmetric encryption

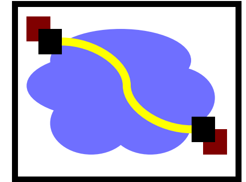
The Dreaded PKI



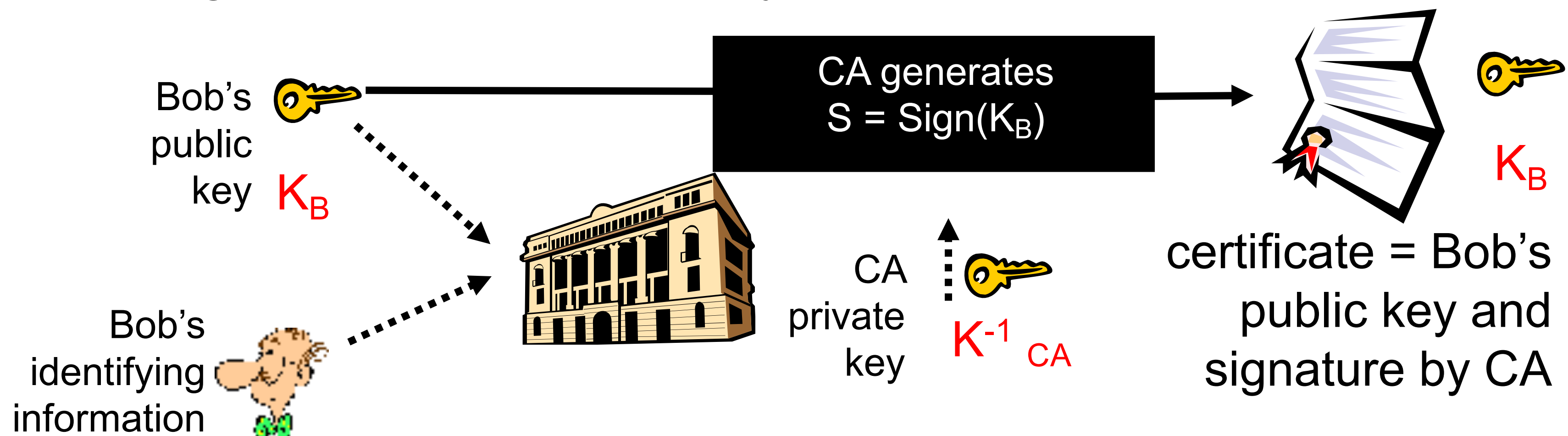
- Definition: Public Key Infrastructure (PKI)

1) A system in which “roots of trust” authoritatively bind public keys to real-world identities
2) A significant stumbling block in deploying many “next generation” secure Internet protocol or applications.

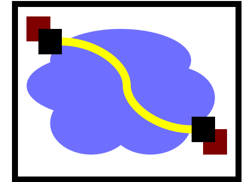
Certification Authorities



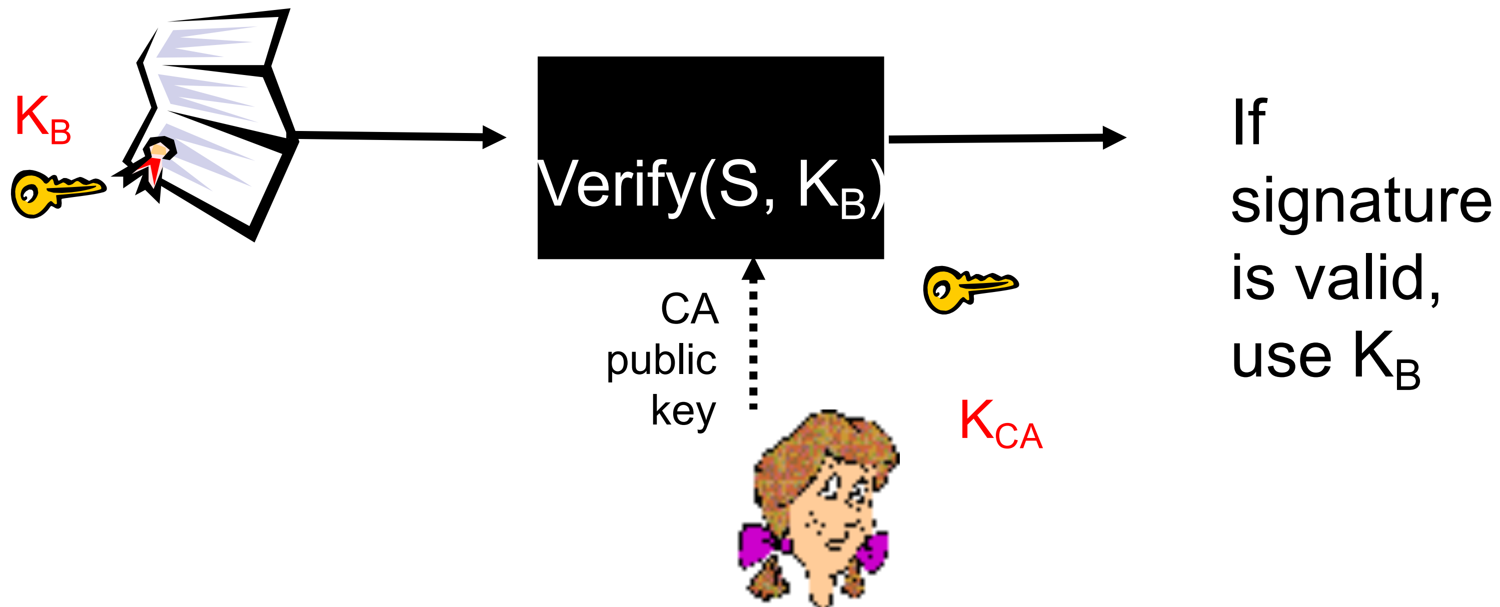
- Certification authority (CA): binds public key to particular entity, E.
- An entity E registers its public key with CA.
  - E provides “proof of identity” to CA.
  - CA creates certificate binding E to its public key.
  - Certificate contains E’s public key AND the CA’s signature of E’s public key.



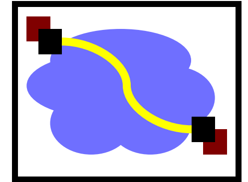
Certification Authorities



- When Alice wants Bob's public key:
  - Gets Bob's certificate (Bob or elsewhere).
  - Use CA's public key to verify the signature within Bob's certificate, then accepts public key



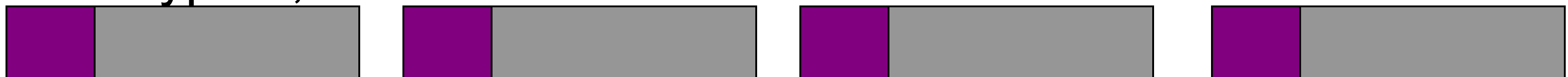
How TLS/SSL Handles Data



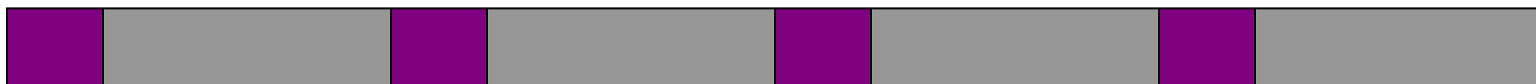
1) Data arrives as a stream from the application via the TLS Socket

2) The data is segmented by TLS into chunks

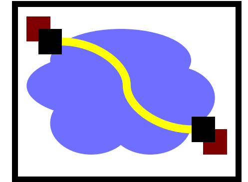
3) A session key is used to encrypt and MAC each chunk to form a TLS “record”, which includes a short header and data that is encrypted, as well as a MAC.



4) Records form a byte stream that is fed to a TCP socket for transmission.

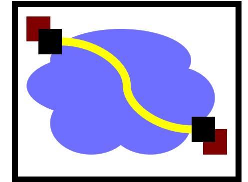


Analysis



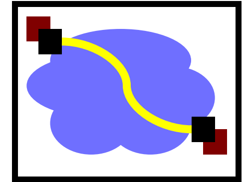
- PKI lets us take the trusted third party offline:
  - If it's down, we can still talk!
  - But we trade-off ability for fast revocation
    - If server's key is compromised, we can't revoke it immediately...
    - Usual trick:
      - Certificate expires in, e.g., a year.
      - Have an on-line revocation authority that distributes a revocation list. Kinda clunky but mostly works, iff revocation is rare. Clients fetch list periodically.
- Better scaling: CA must only sign once... no matter how many connections the server handles.
- If CA is compromised, attacker can trick clients into thinking they're the real server.

Forward secrecy



- In KDC design, if key $K_{server\text{-}KDC}$ is compromised a year later,
  - from the traffic log, attacker can extract session key (encrypted with auth server keys).
  - attacker can decode all traffic retroactively.
- In SSL, if CA key is compromised a year later,
  - Only new traffic can be compromised. Cool…
- But in SSL, if server's key is compromised...
  - Old logged traffic can still be compromised...

Diffie-Hellman Key Exchange



- Different model of the world: How to generate keys between two people, securely, no trusted party, even if someone is listening in.

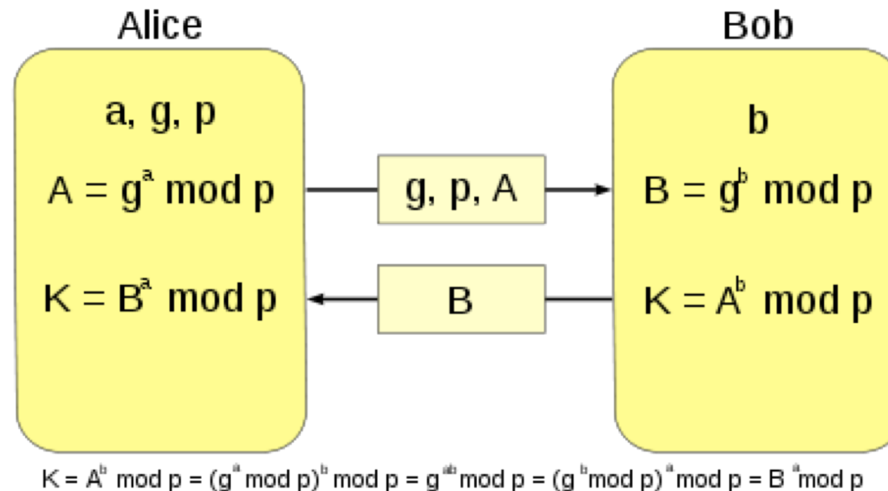
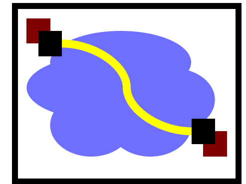
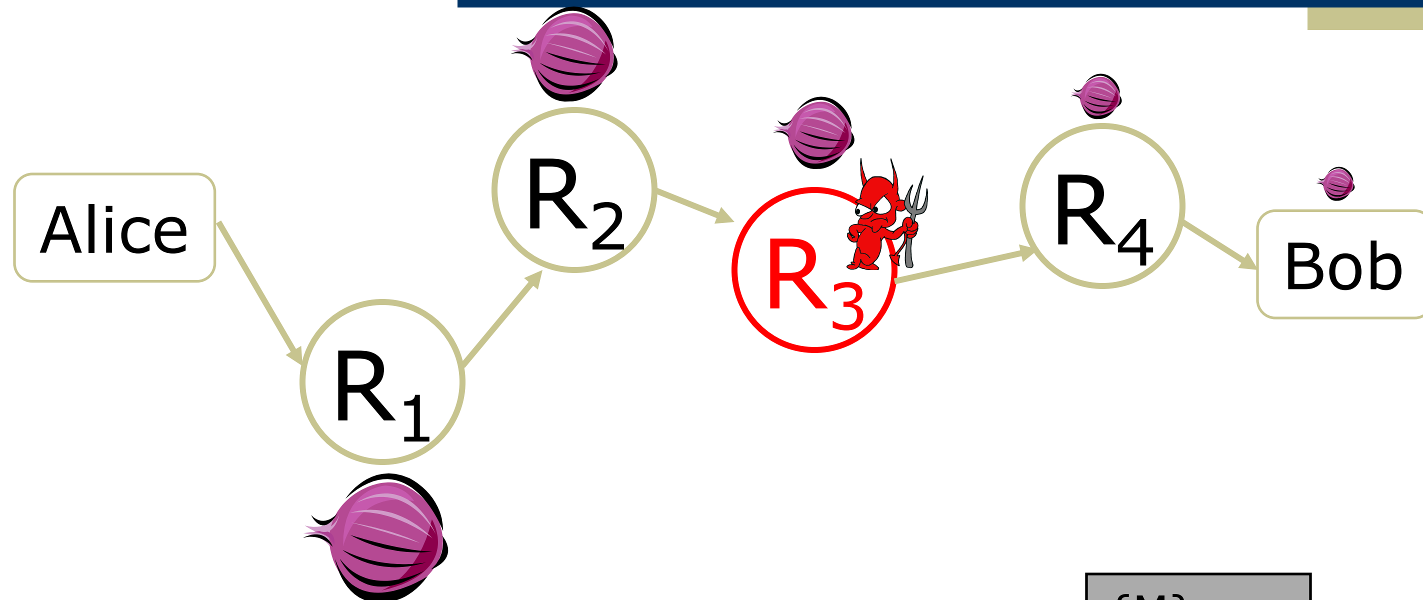


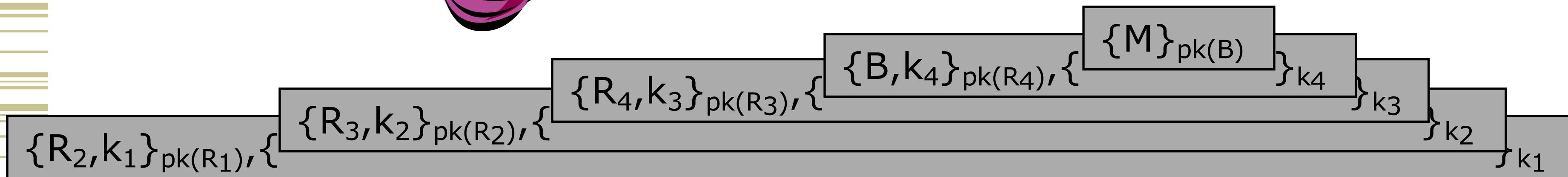
image from wikipedia

- This is cool. But: Vulnerable to man-in-the-middle attack. Attacker pair-wise negotiates keys with each of A and B and decrypts traffic in the middle. No authentication...

Overall Route Establishment





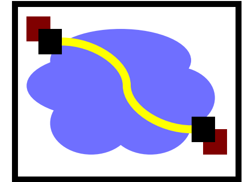


Routing info for each link encrypted with router's public key

Each router learns only the identity of the next router

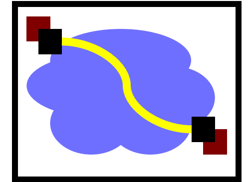
Note: k_1, k_2, k_3 etc are session keys, so when each router (R_1, R_2, .. R_n) use their private keys to decrypt the packets, they can only then get the next hop (e.g. R_2) and the session key (k_1) to decrypt the rest of the packet and send it along.

Authentication?

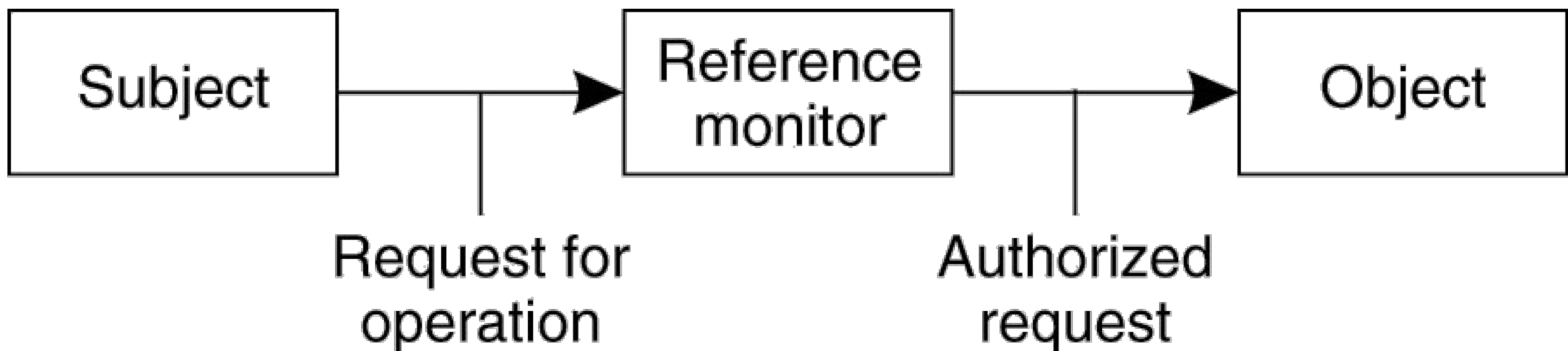


- But we already have protocols that give us authentication!
  - They just happen to be vulnerable to disclosure if long-lasting keys are compromised later...
- Hybrid solution:
  - Use diffie-hellman key exchange with the protocols we've discussed so far.
  - Auth protocols prevent M-it-M attack if keys aren't yet compromised.
  - D-H means that an attacker can't recover the real session key from a traffic log, even if they can decrypt that log.
  - Client and server discard the D-H parameters and session key after use, so can't be recovered later.
- This is called "perfect forward secrecy". Nice property.

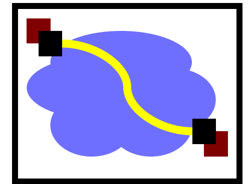
Access Control



- Once secure communication between a client and server has been established, we now have to worry about access control – when the client issues a request, how do we know that the client has **authorization**?



The Access Control Matrix (ACM)

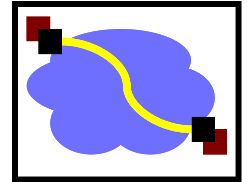


A model of protection systems

- Describes who (subject) can do what (rights) to what/whom (object/subject)
- Example
  - An instructor can assign and grade homework and exams
  - A TA can grade homework
  - A Student can evaluate the instructor and TA

33

Two ways to cut a table (ACM)



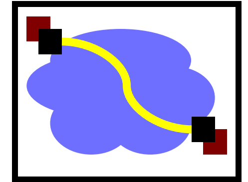
- Order by columns (ACL) or rows (Capability Lists)?

	File1	File2	File3
Ann	*rx*	*r*	rwx
Bob	*rwx*	r	--
Charlie	*rx*	rw	w

ACLs

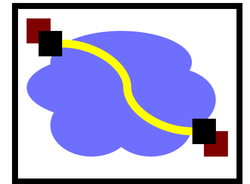
Capability

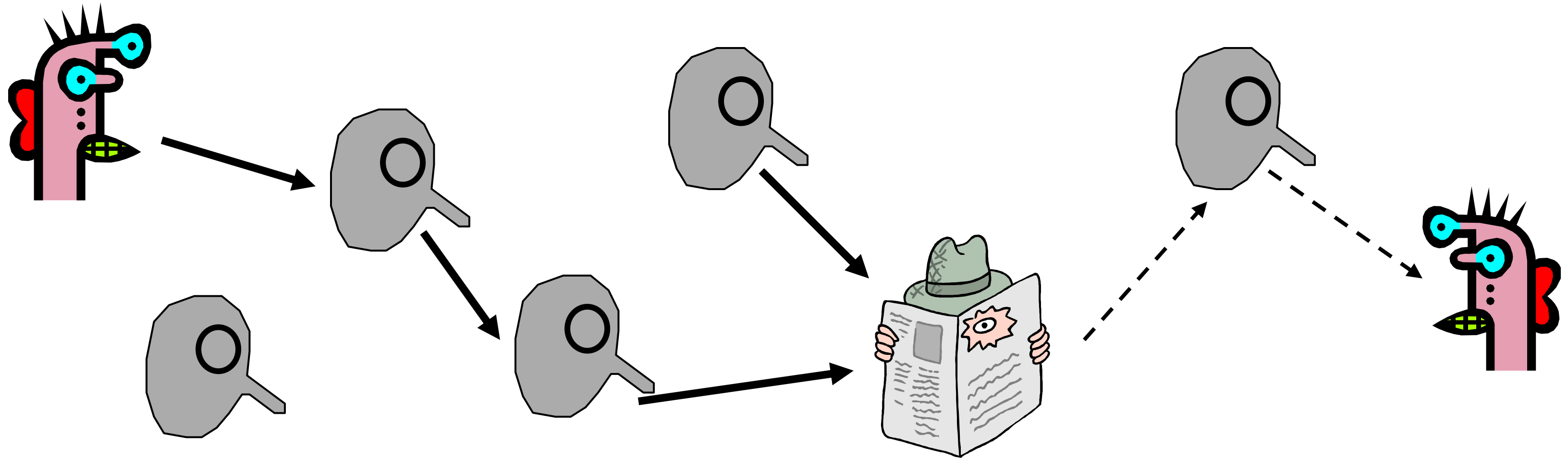
ACLs vs. Capabilities



- They are equivalent:
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
  - ACLs answer second easily; C-Lists, answer the first easily.
- The second question in the past was most used; thus ACL-based systems are more common
- But today some operations need to answer the first question

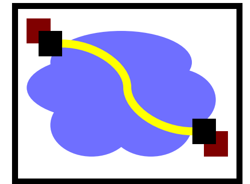
Randomized Routing





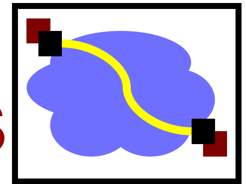
- Hide message source by routing it randomly
  - Popular technique: Crowds, Freenet, Onion routing
- Routers don't know for sure if the apparent source of a message is the true sender or another router

Onion Routing



- Sender chooses a random sequence of routers
  - Some routers are honest, some controlled by attacker
  - Sender controls the length of the path

IP Layering & Encryption Protocols



	Host	Bridge/Switch or a WiFi AP	Router/Gateway	Host
Application				
Transport				
Network				
Link				
Physical				

SSL/TLS

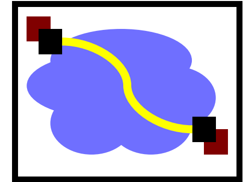
IPSec

802.1x, …
WPA/WEP
For WiFi

So, what does using encrypted WiFi protect against?

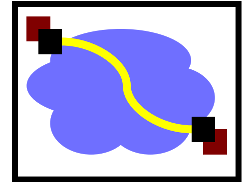
…. How about SSL to google.com on Starbucks open WiFi?

Key Bits: Today's Lecture



- Effective secure channels
  - Key Distribution Centers and Certificate Authorities
  - Diffie-Hellman for key establishment in the “open”
- Access control
  - Way to store what “subjects” can do to “objects”
  - Access Control Matrix: ACLs and Capability lists
- Privacy and Tor
  - Used for anonymity on the internet (Onion Routes)
  - Uses ideas from encryption, networking, P2P

One Final Logistical Update!



- Please fill out course evaluations (FCE)
  - Helps us improve the course, we appreciate feedback
- We will use the last 5 mins of class today for this
  - Daniel and I will step out to not influence you ☺

Thank You!