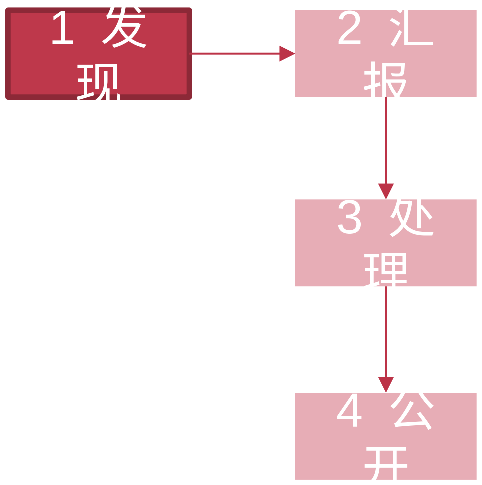
SE3331-1 (2023 Fall)

Privacy & Review

IPADS, Shanghai Jiao Tong University

https://www.sjtu.edu.cn

Process of fixing a bug/vulnerability



Based on a real case (Credit: Fan Yang)

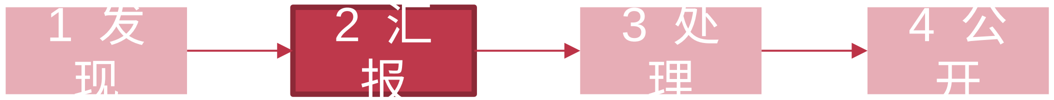
- CVE-2020-10757
- Linux commit 5c7fb56e5e3f

Phase-1: finding a bug

- Survey whether the bug has been found
- Simply the process of re-producing bug
- Evaluate the seriousness, if belongs to security, offer an exploit
- Provide a patch if you have one.

TCB: Yourself

Fix the Bug



Phase-2: Report

– Not security bug： kernel mailing list, bugzilla *public*

– Security bug： security@kernel.org *Non-public*

– Require a CVE id (Common Vulnerability and Exposures List)

- Not public yet
- CVE status is RESERVED

TCB: Mailing list + Yourself

Fix the Bug

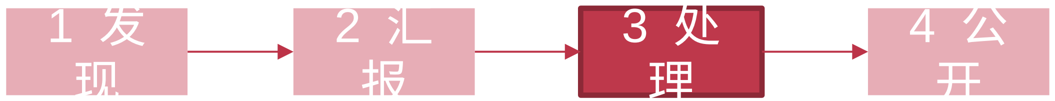
$$\begin{tikzcd}\text{1 发现} \arrow[r] & \text{2 汇报} \arrow[r] & \boxed{\text{3 处理}} \arrow[r] & \text{4 公开}\end{tikzcd}$$

Phase-3: handle the bug

– Related developers and maintainers join the discussion

TCB: Related developers + Mailing list + Yourself

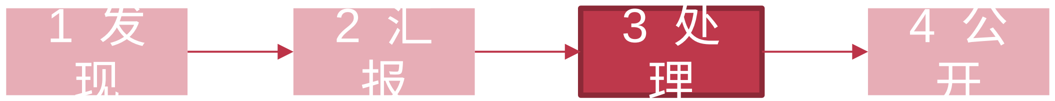
Fix the Bug



Phase-3: handle the bug

– Patch proposing and discussion

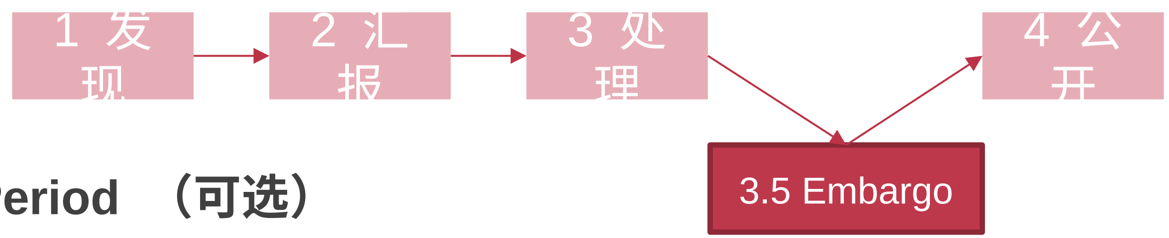
Fix the Bug



Phase-3: handle the bug

– Test and patch review

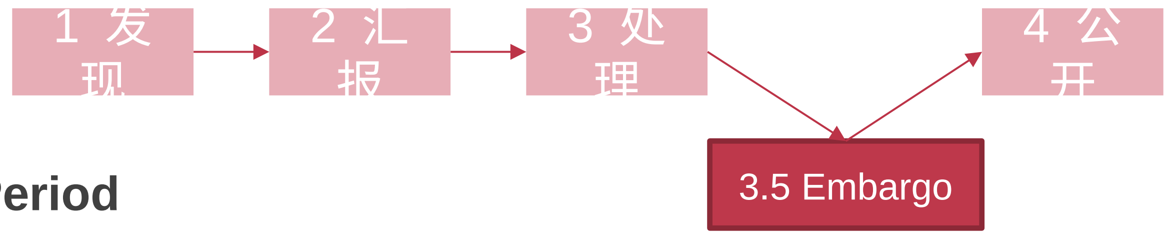
Fix the Bug



Phase-3.5： Embargo Period （可选）

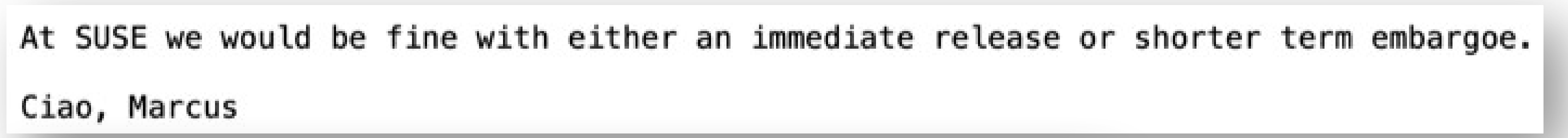
- discuss with maintainers of distributions, prepare to fix
- Why not just fix it?
  - Submit a patch in public mailing list means public
  - Prevent distributions not fix the bugs in time
- Decide whether embargo period is needed: evaluate

Fix the Bug

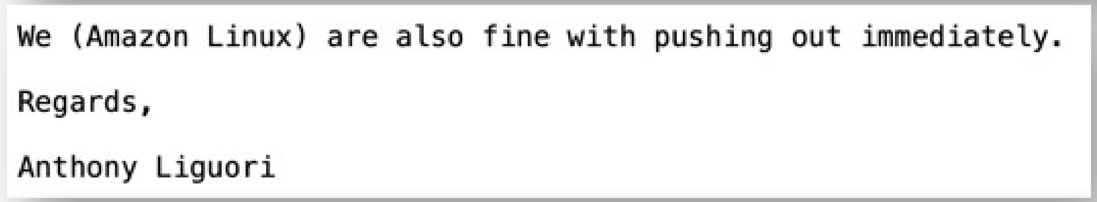


Phase-3.5： Embargo Period

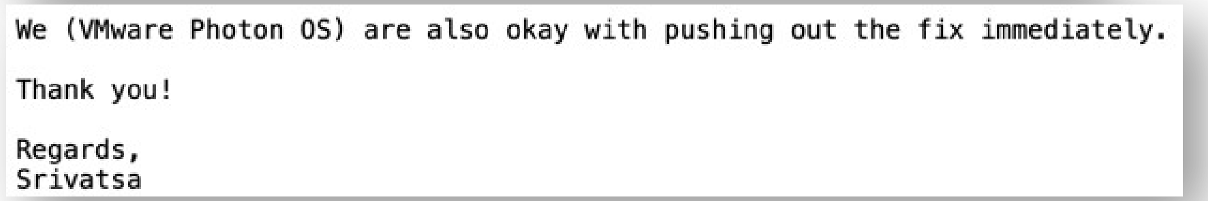
– You can negociate Embargo Period



```
At SUSE we would be fine with either an immediate release or shorter term embargoe.

Ciao, Marcus
```



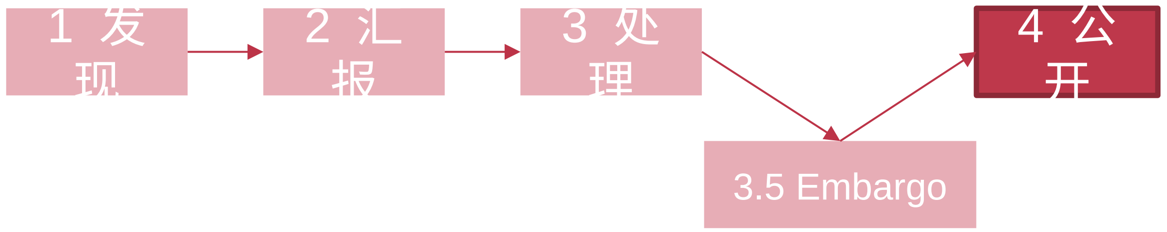
```
We (Amazon Linux) are also fine with pushing out immediately.

Regards,

Anthony Liguori
```



```
We (VMware Photon OS) are also okay with pushing out the fix immediately.

Thank you!

Regards,
Srivatsa
```

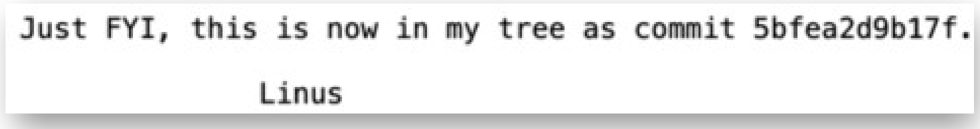
TCB: Distributions + Related developers + Mailing list + Yourself

Fix the Bug



Phase-4: Public

- Merge to mainstream



```
Just FYI, this is now in my tree as commit 5bfea2d9b17f.

               Linus
```

- Backport to related stable versions
- Publish to public mailing list, security group
  - E.g., oss-security@lists.openwall.com
- Update CVE status to public

TCB: All!

PRIVACY

Data Privacy

Data can be used everywhere

- Risk management
- Medicine
- Recommended system
- …

Data can be stolen easily

- Everyone who uses your data can steal it

Data Privacy

What’s the target of data privacy system?

- Allow data **to be used**, and **protect it from being stolen**

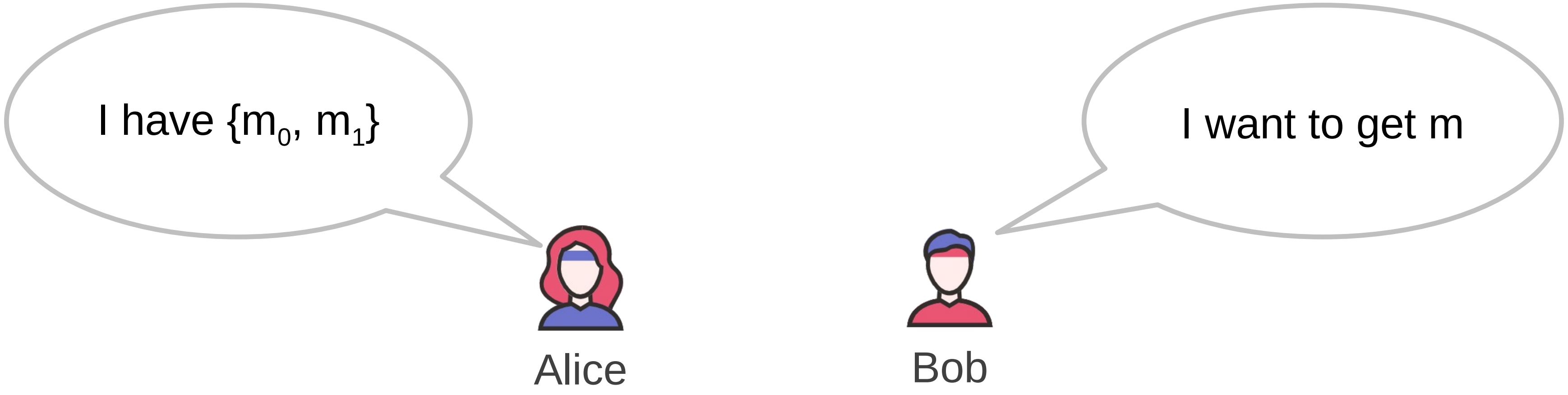
Today's contents

- Basic data privacy method
  - ZKP, OT, HE, sMPC, TEE, DP
- Systems which try to enforce data privacy

Oblivious Transfer

OT

Problem



- Alice has $\{m_0, m_1\}$ and Bob wants to get m , $\in\{0,1\}$
- Alice may know the **m**
- Bob may get both $\mathbf{m_0}$ and $\mathbf{m_1}$

Oblivious Transfer (OT)

- Introduced by Michael O. Rabin, 1981
- Scenario: message transfer
  - A sender has a message list **{m_0, m_1, … , m_n}**
  - A receiver wants to get **k** target messages from sender
- Properties: oblivious and secure
  - Oblivious: sender **cannot know which messages are received**
  - Secure: receiver can **only get the target messages**

1-out-of-2 OT (one solution)

Scenario: **RSA encryption**

$\{m_0, m_1\}$

$m = ?$

Alice

Bob

Generate 2 key pairs

$\{K0_{pub}, K0_{prv}\}, \{K1_{pub}, K1_{prv}\}$

$\xrightarrow{K0_{pub},\ K1_{pub}}$

Generate a random number **r**

$c \leftarrow \mathrm{Enc}(K\sigma_{pub}, r)$

$\xleftarrow{c}$

$k_0 \leftarrow \mathrm{Dec}(K0_{prv}, c)$

$k_1 \leftarrow \mathrm{Dec}(K1_{prv}, c)$

$e_0 \leftarrow k_0 \oplus m_0$

$e_1 \leftarrow k_1 \oplus m_1$

$\xrightarrow{e_0,\ e_1}$

$m_\sigma \leftarrow e_\sigma \oplus r$

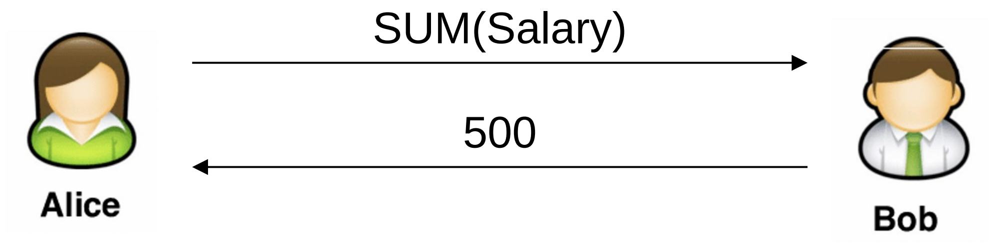
More OT Protocols

- Different numbers of selected messages
    - 1-out-of-2 OT
    - 1-out-of-n OT
    - k-out-of-n OT
- Implementation method
    - Non-adaptive OT
    - Adaptive OT
    - Publicly Verifiable OT
    - …

Differential privacy

DP

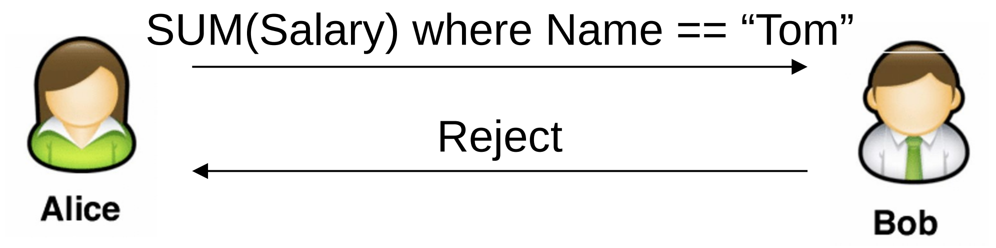
Problem



Name	Salary
Alice	100
Bob	80
Brown	200
Tom	120

- Alice can perform queries on Bob’s database, but cannot access a single database entry

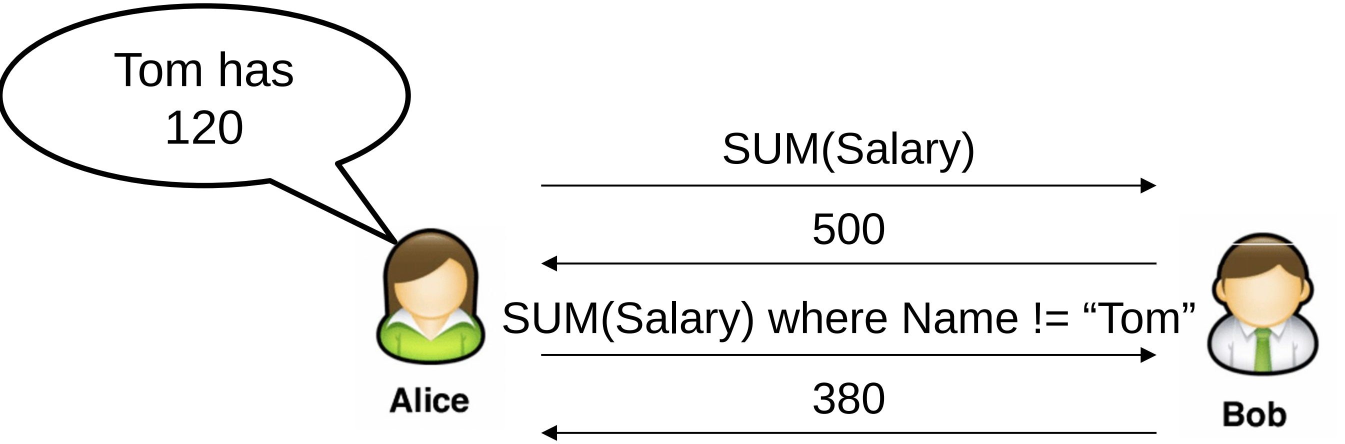
Problem



Name	Salary
Alice	100
Bob	80
Brown	200
Tom	120

- Alice can perform queries on Bob’s database, but cannot access a single database entry
  - Naïve method: reject Alice to access single entry

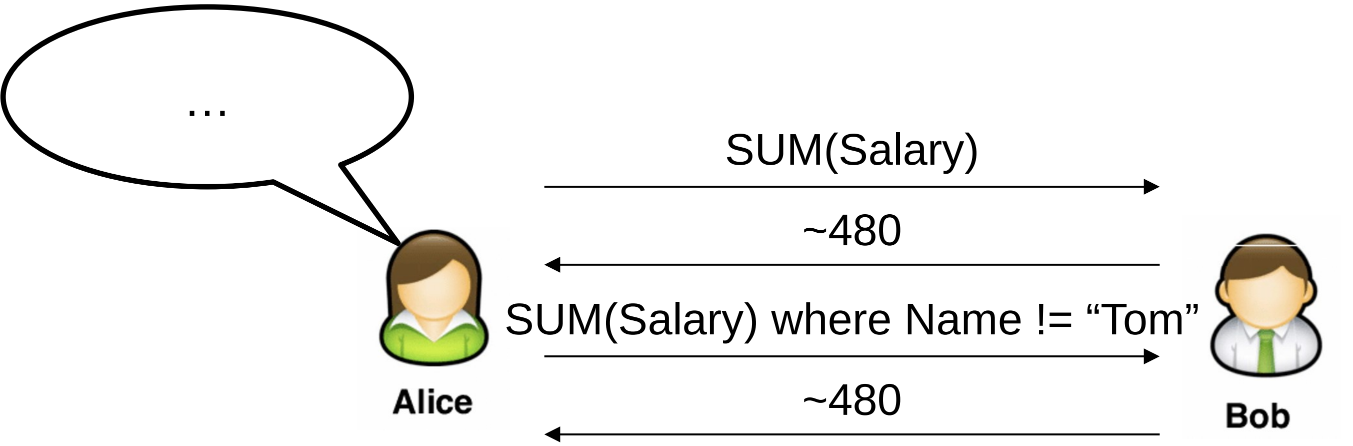
Problem



Name	Salary
Alice	100
Bob	80
Brown	200
Tom	120

- Alice can perform queries on Bob's database, but cannot access a single database entry
  - Naïve method: reject Alice to access single entry

When DP is Enabled



Name	Salary
Alice	100
Bob	80
Brown	200
Tom	120

SECRET SHARING

Secret Sharing

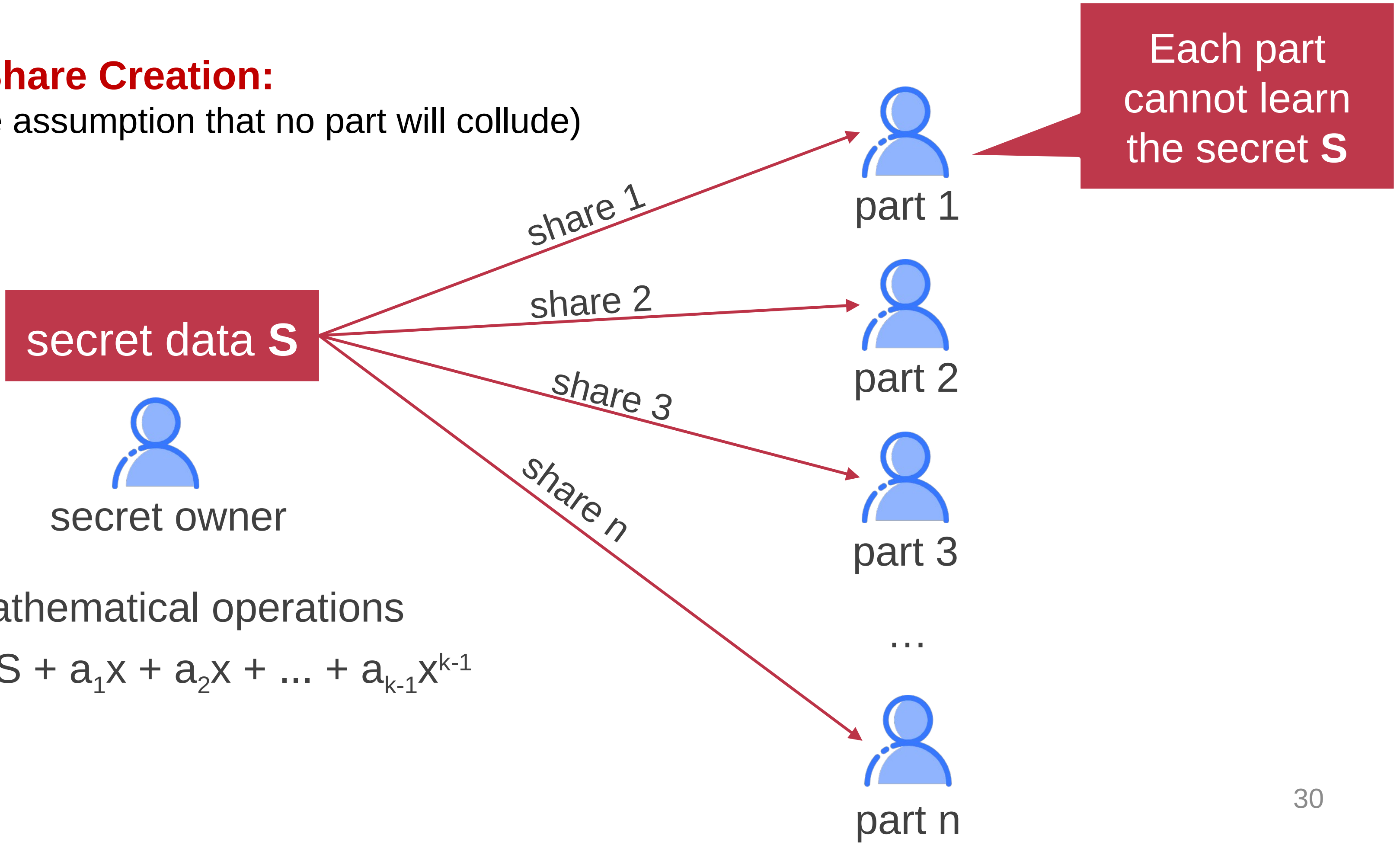
Introduced by Adi Shamir, 1979

Key Insight: One point constructs infinite number of lines, two points construct only one line

Key Idea: Split a secret data into *n* shares, at least *k* shares can reconstruct the secret ((*k*,*n*)-SS)

Secret Sharing

Secret-Share Creation:
(under the assumption that no part will collude)

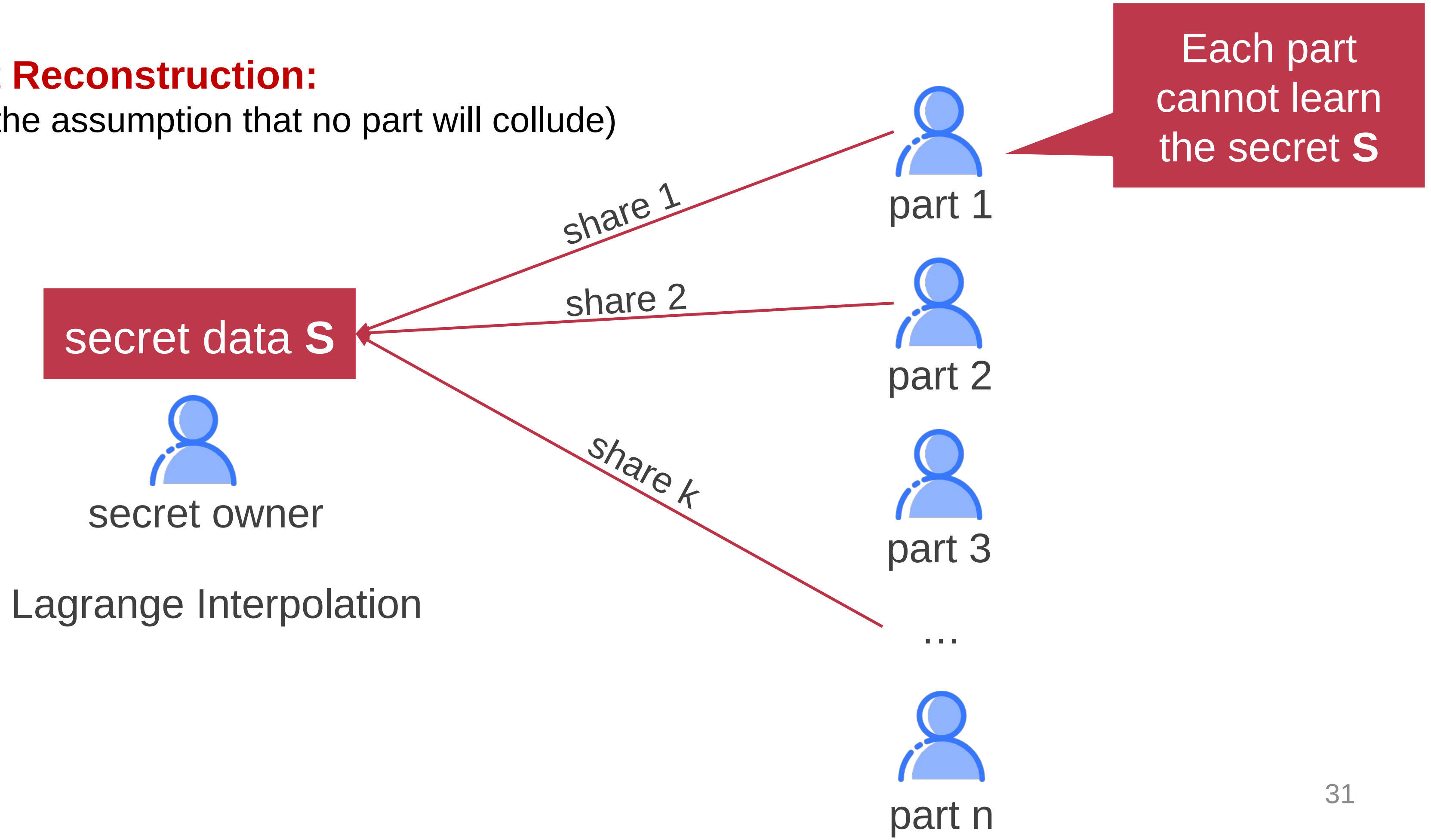


Mathematical operations

$f(x) = S + a_1x + a_2x + ... + a_{k-1}x^{k-1}$

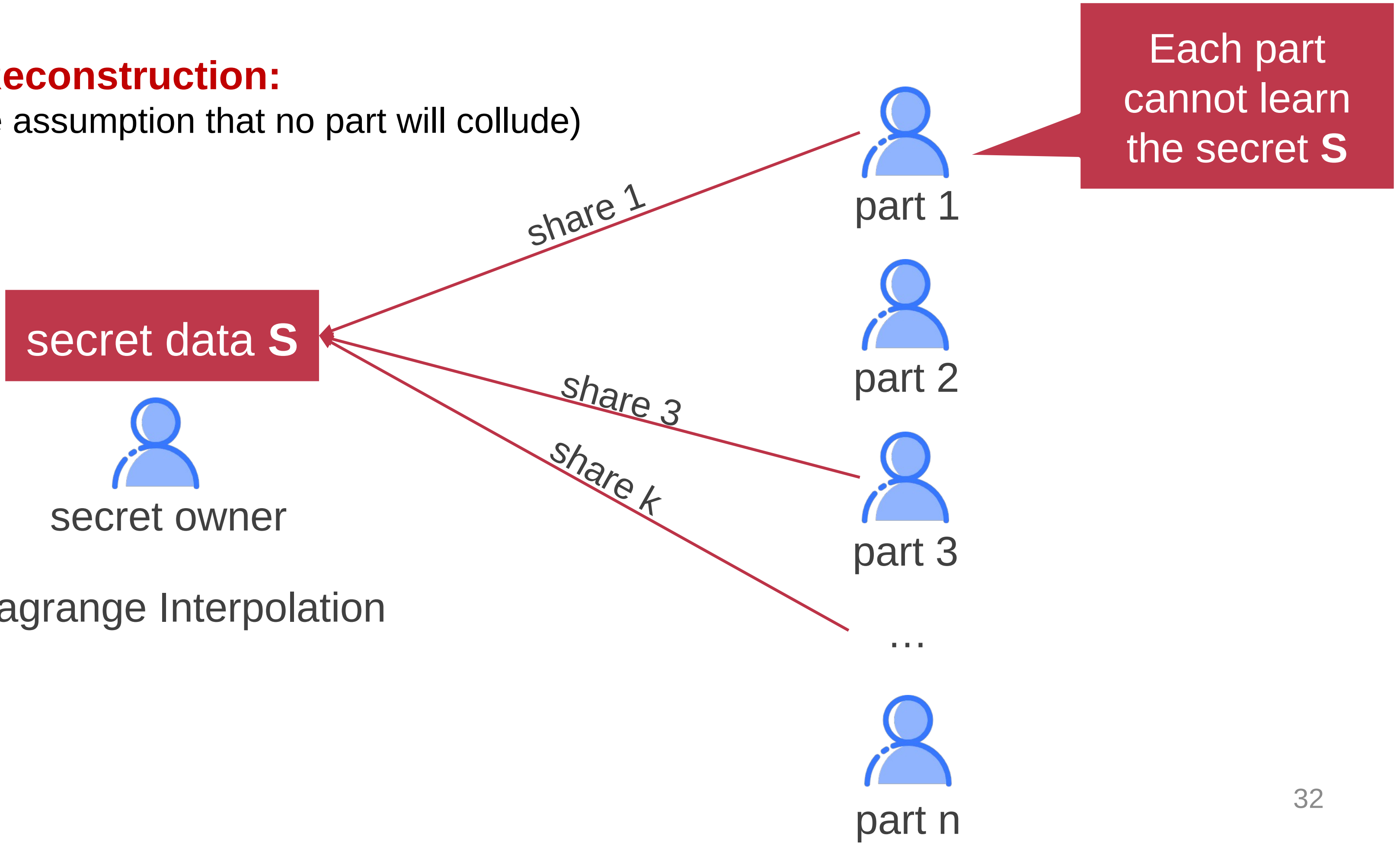
Secret Sharing

Secret Reconstruction:
(under the assumption that no part will collude)



Secret Sharing

Secret Reconstruction:
(under the assumption that no part will collude)



Lagrange Interpolation

Pros and Cons

An information-theoretically secure protocol (unconditionally secure, independent of adversary's computational capabilities → quantum-safe)

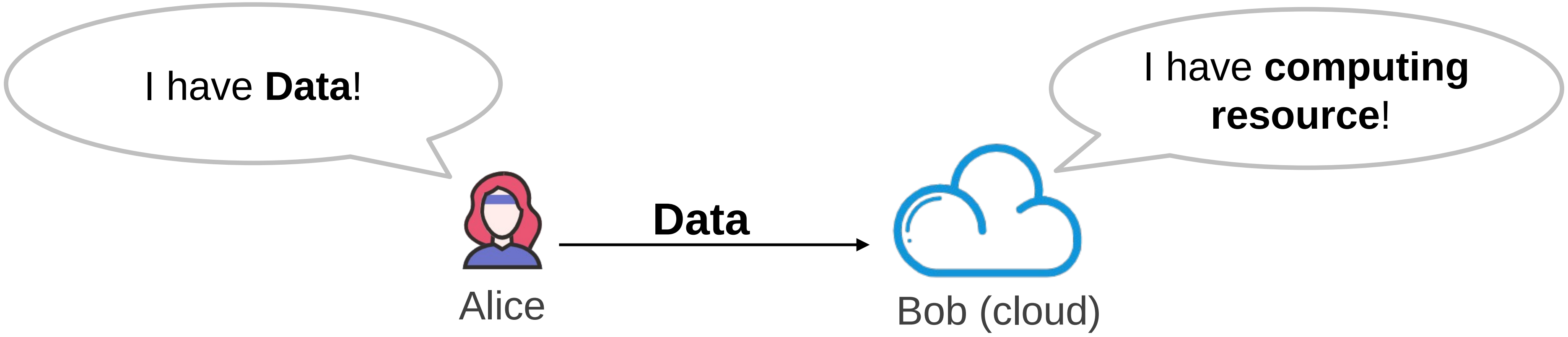
Ideal assumption (At least *k* parts never collude)

Communication overhead

Homomorphic Encryption

HE

Problem

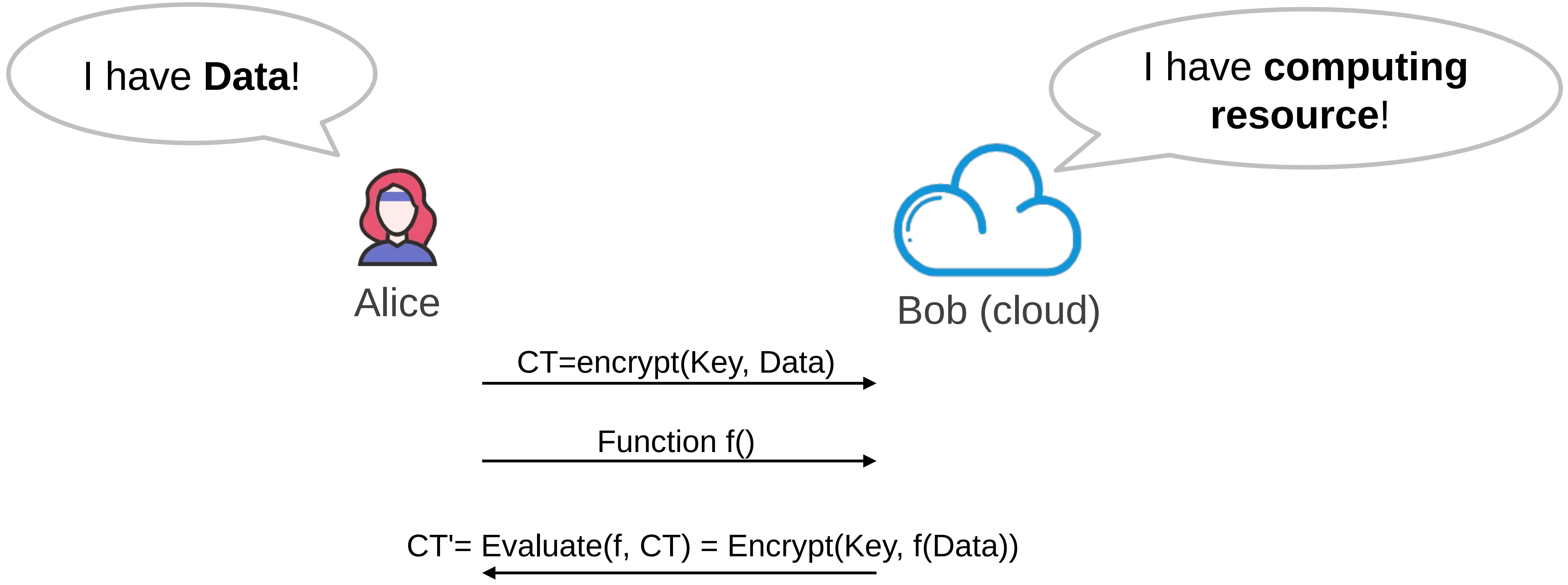


Alice wants to ask Bob (e.g., a cloud) to perform calculation on her data

Naïve method: Sending Data to Bob

- Bob will get the data

Homomorphic Encryption (HE)


I have **Data**!
I have **computing resource**!
Alice
Bob (cloud)
CT=encrypt(Key, Data)
Function f()
CT'= Evaluate(f, CT) = Encrypt(Key, f(Data))

Get f(Data) = Decrypt(Key, CT')

SWHE and FHE

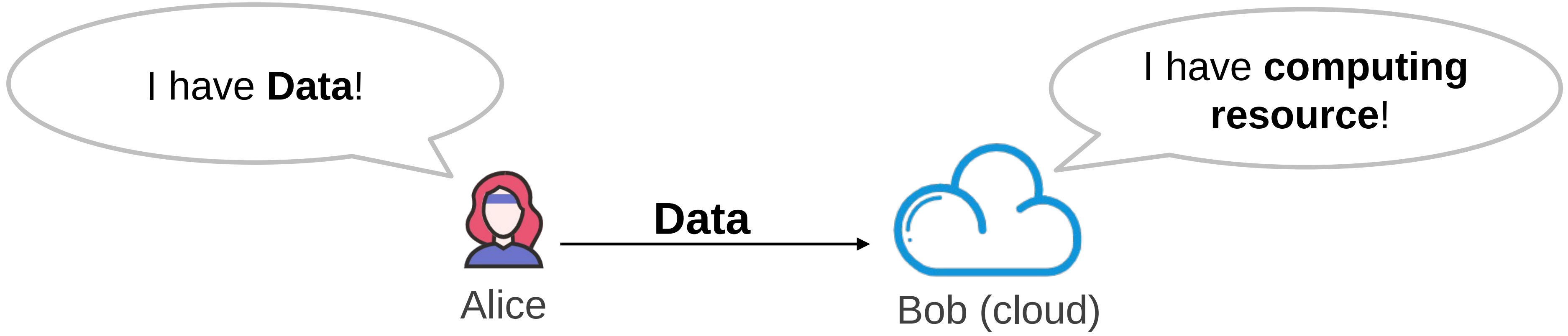
- HE: Homomorphic Encryption

- SWHE: SomeWhat Homomorphic Encryption
  - Support **limited** kinds and times of operation
  - (e.g., RSA)
  - (e.g., Paillier)
- FHE: Full Homomorphic Encryption
  - Support all kinds of operations
  - Addition and multiplication

Trusted Execution Environment

TEE

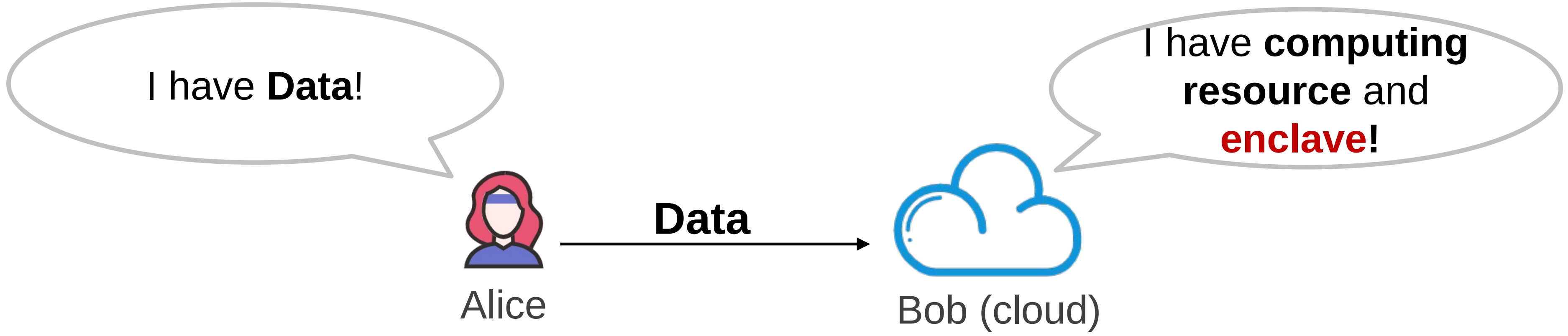
Problem



Alice wants to ask Bob (e.g., a cloud) to perform calculation on her data

Naïve method: Sending Data to Bob

Trusted Execution Environment (TEE)



Alice wants to ask Bob (e.g., a cloud) to perform calculation on her data

~~Naïve method: Sending Data to Bob~~

Bob cloud construct an enclave

TEE: Trusted Execution Environment

Widely deployed on mobile devices

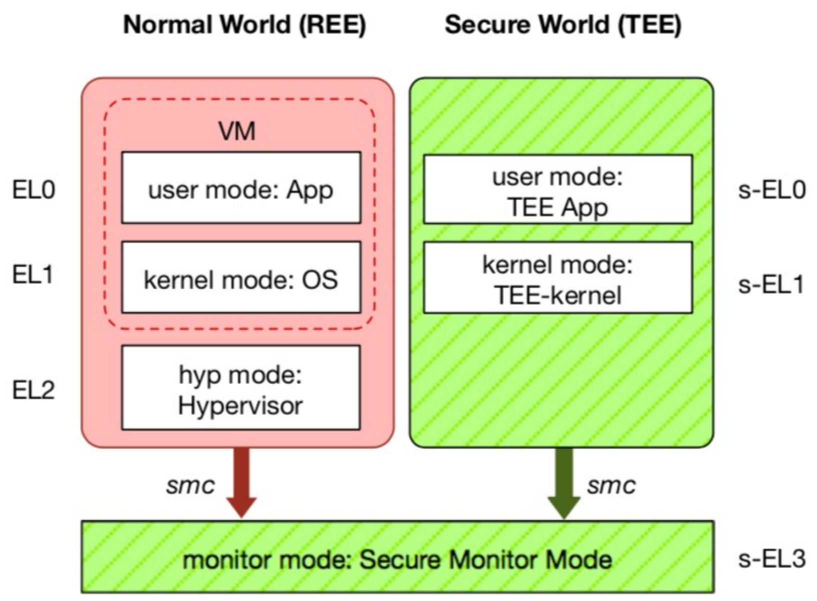
- Every Android device must deploy TEE
- Mostly based on ARM TrustZone

An *isolated* & *privileged* environment

- TEE can access all DRAM & peripherals
- REE cannot access TEE's resources

Designed to be secured and small

- For security-critical tasks (e.g., fingerprint checker, mobile payment, etc.)



Two Features of TEE

1. Isolated execution

- Minimal TCB: system software is not trusted
  - E.g., OS and hypervisor
- Some can prevent physical attacks
  - With memory encryption

2. Remote attestation

- Prove itself to the end users
- Usually use SSL to establish a secure channel over network

Different TEEs

- Software TEE
  - VM-based TEE
  - Same privilege protection
- ARM TrustZone
- Intel SGX
- AMD SME/SEV
- SANCTUM

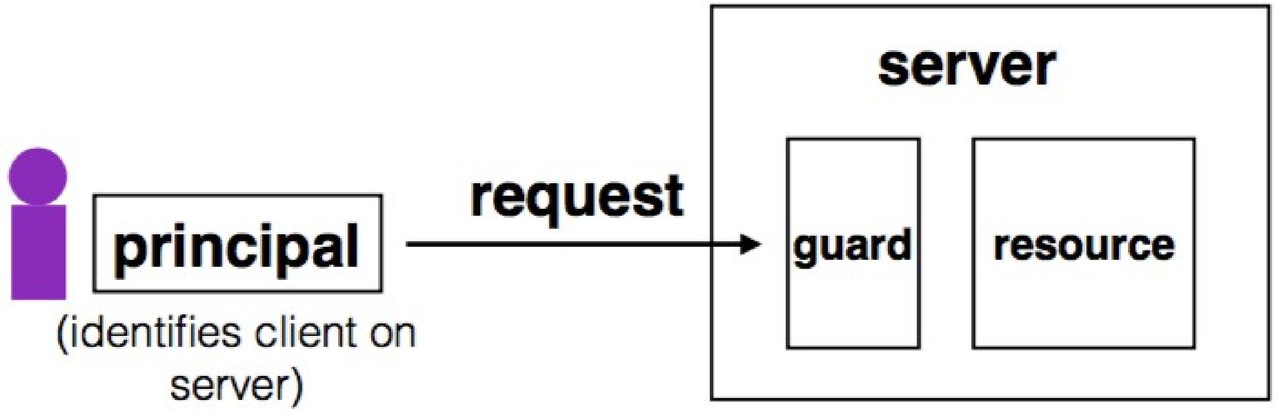
Review: Security

2 Steps towards Building a More Secure System

How to make progress in building a secure system?

- Be clear about goals: "**policy**"
- Be clear about assumptions: "**threat model**"

Complete Mediation

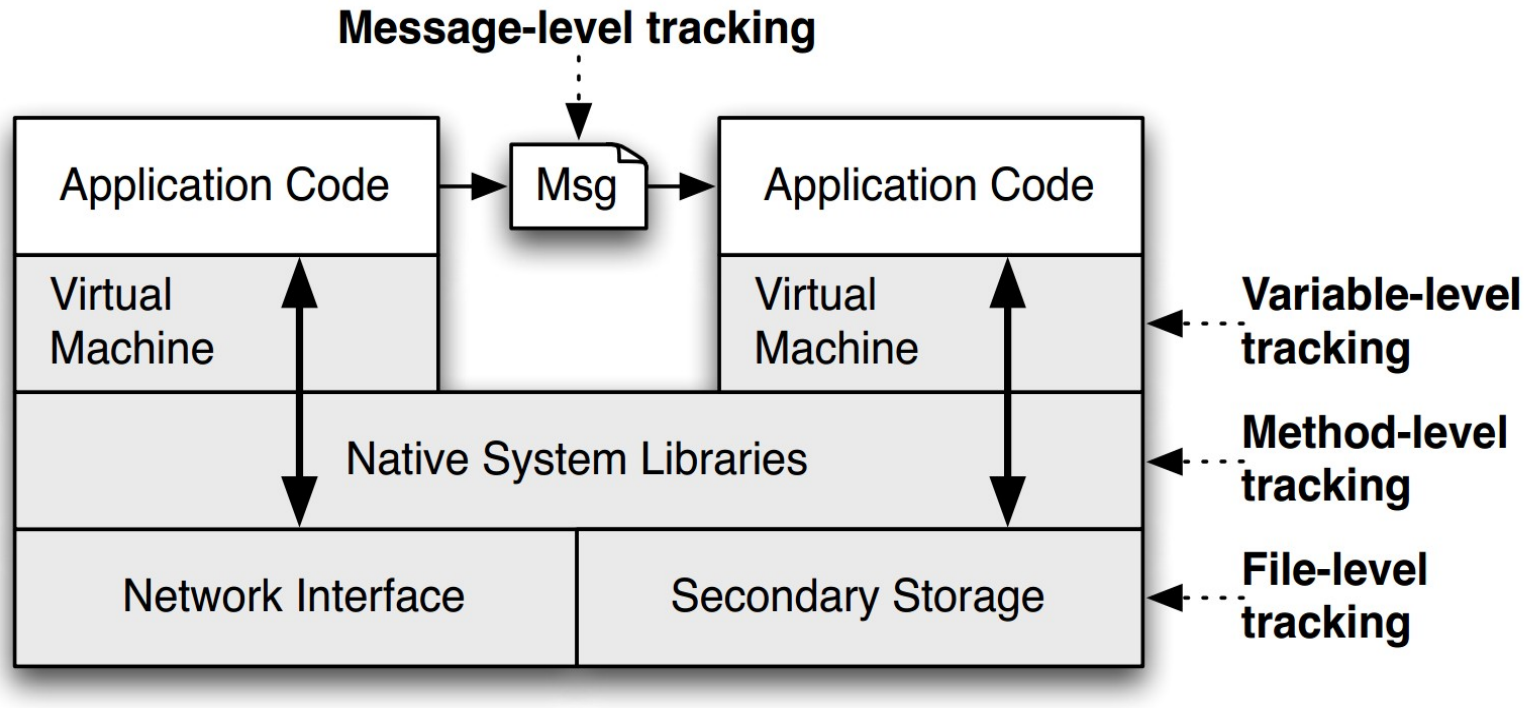


Guard typically provides:

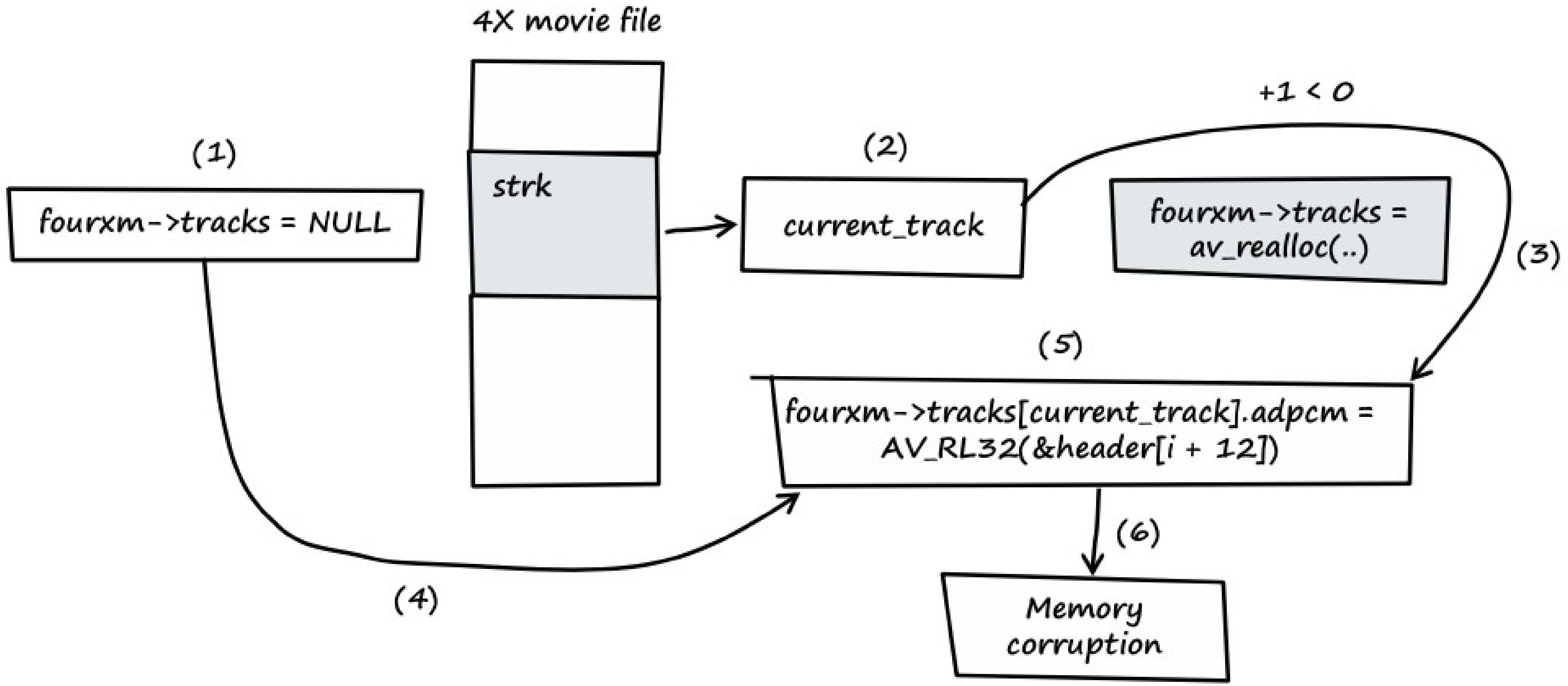
– **Authentication**: is the principal who they claim to be?

– **Authorization**: does principal have access to perform request on resource?

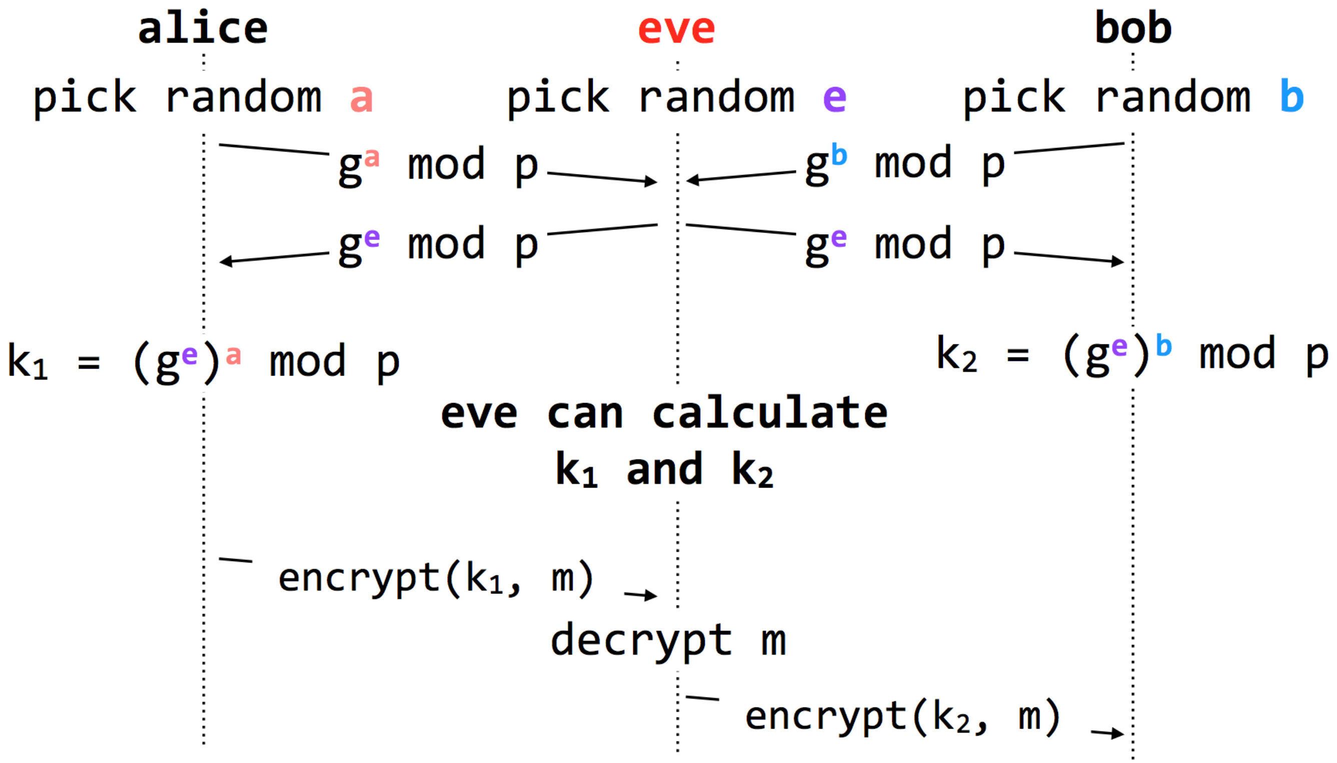
TaintDroid


Message-level tracking
Application Code
Msg
Application Code
Virtual Machine
Virtual Machine
Variable-level tracking
Native System Libraries
Method-level tracking
Network Interface
Secondary Storage
File-level tracking

Trace the Input Data


4X movie file
strk
(1)
fourxm->tracks = NULL
(2)
current_track
+1 < 0
fourxm->tracks = av_realloc(..)
(3)
(5)
fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12])
(4)
(6)
Memory corruption

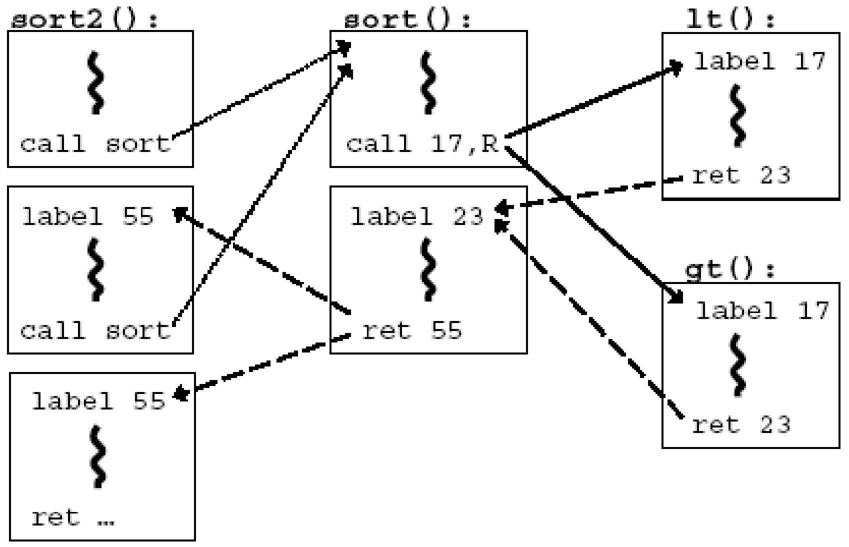
Diffie-Hellman Key Exchange & Man-in-the-Middle



problem: alice and bob don't know they're not communicating directly

CFI: Control Flow Integrity

```
bool lt(int x, int y) {
    return x < y;
}

bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len)
{
    sort( a, len, lt );
    sort( b, len, gt );
}
```

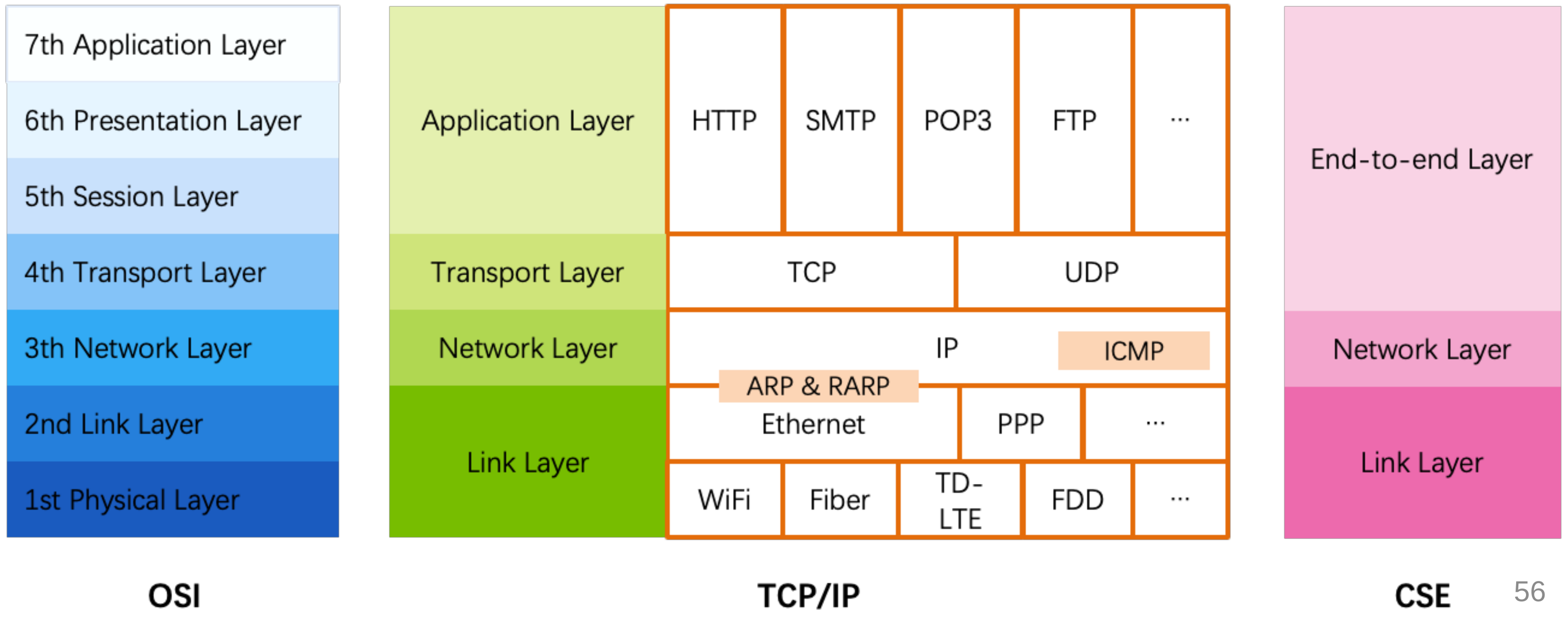


Review: Network

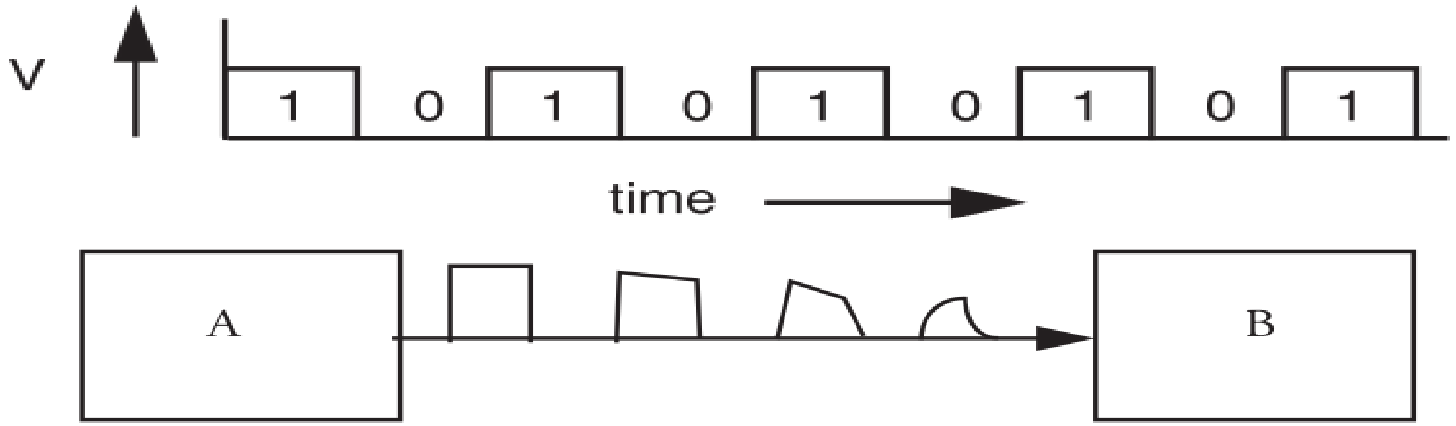
OSI, TCP/IP & Protocol Stack

OSI

- The open systems interconnection (OSI) model
- 7-layer architecture



Signal Transmission on Analog Line



It is hard for B to understand the signal

– B doesn't have a copy of A's clock, so when to sample the signal?

VCO: Voltage Controlled Oscillator

How to make two ends agree on the data rate without clock line?

The receiver run a VCO at about the same data rate

- VCO's output is multiplied by the voltage of incoming signal
- The product is suitably filtered and sent back to adjust the VCO
- VCO will finally be **locked** to both the frequency and phase of the arriving signal: phase-locked loop
- Then the VCO becomes a clock source for the receiver

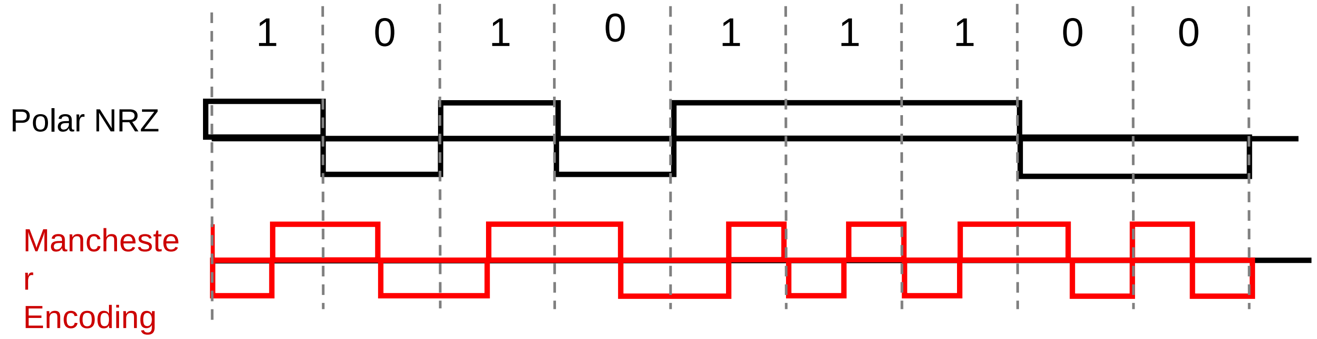
Problem: if no transition in the stream (e.g., a lot of zero), the phase-locked loop cannot synchronize

Manchester Code

Solution: sender encodes the data to ensure transitions

Phase encoding: at least 1 level transition for a bit

- Manchester code: 0 -> 01, 1 -> 10
- Max data rate is only half, but simple enough



Example-2: 4-bit -> 7-bit

4 bits -> 7 bits (56 using only extra 7)

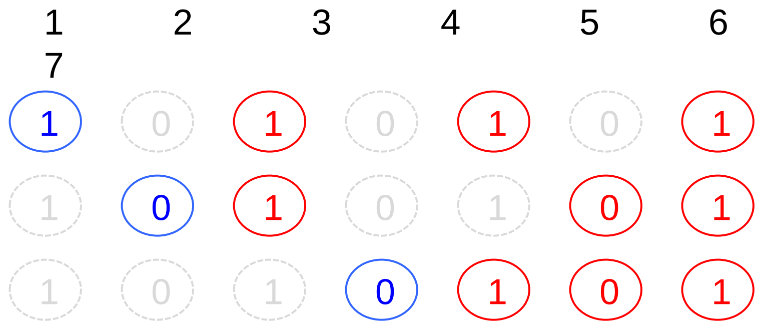
- 3 extra bits to distinguish 8 cases
- e.g. 1101 -> 1010101

$$P_1 = P_7 \oplus P_5 \oplus P_3$$
$$P_2 = P_7 \oplus P_6 \oplus P_3$$
$$P_4 = P_7 \oplus P_6 \oplus P_5$$

Correct 1-bit errors

- 1010101 -> 1010<u>0</u>01 : P1 & P4 not match
- 1010101 -> 1<u>1</u>10101 : P2 not match

Not Match	Error
None	None
P1	P1
P2	P2
P4	P4
P1 & P2	P3
P1 & P4	P5
P2 & P4	P6
P1 & P2 & P4	P7



Control-plane VS. Data-plane

Control-plane

- Control the data flow by defining rules
- E.g., the routing algorithm

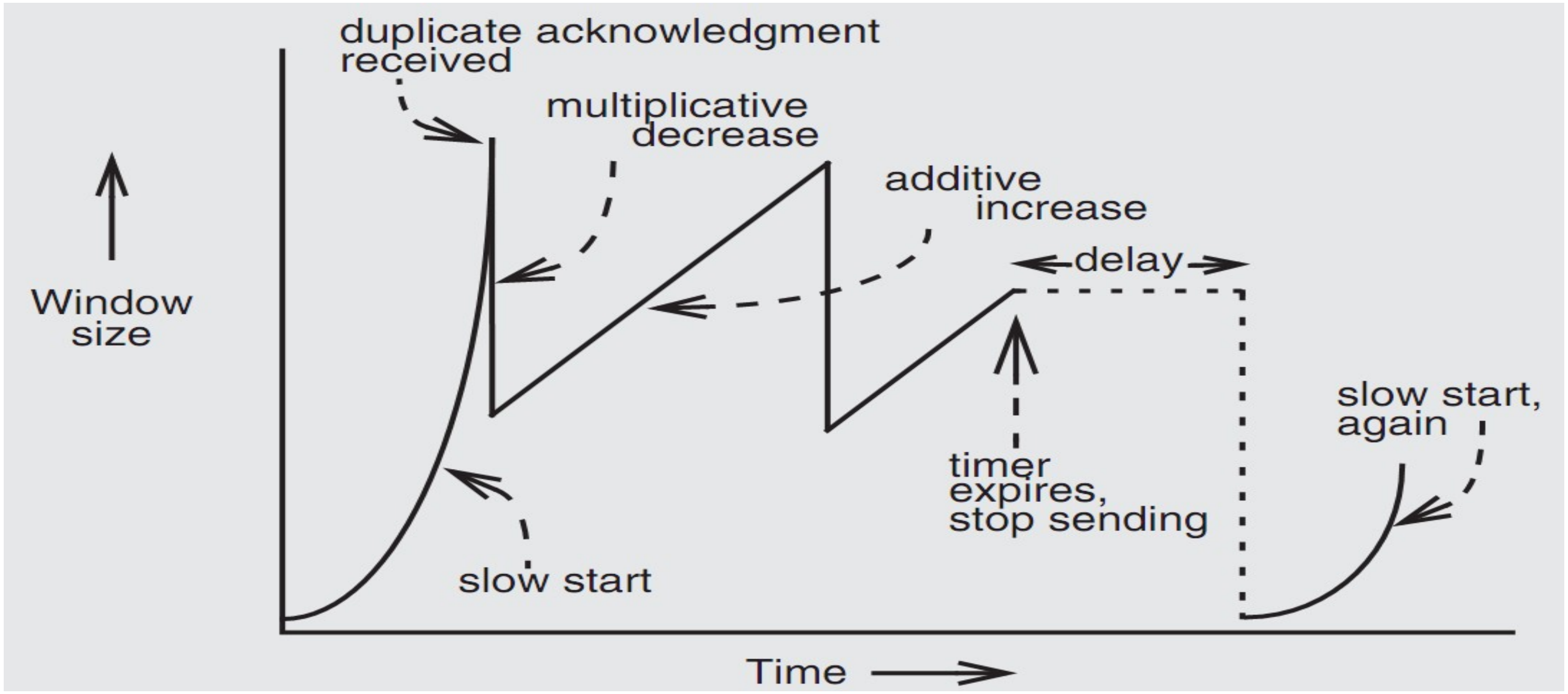
Data-plane

- Copies data according to the rules
- Performance critical
- E.g., the IP forwarding process

Assurance of End-to-end Protocol

1. **Assurance of at-least-once delivery**
2. **Assurance of at-most-once delivery**
3. **Assurance of data integrity**
4. **Assurance of stream order & closing of connections**
5. **Assurance of jitter control**
6. **Assurance of authenticity and privacy**
7. **Assurance of end-to-end performance**

Retrofitting TCP

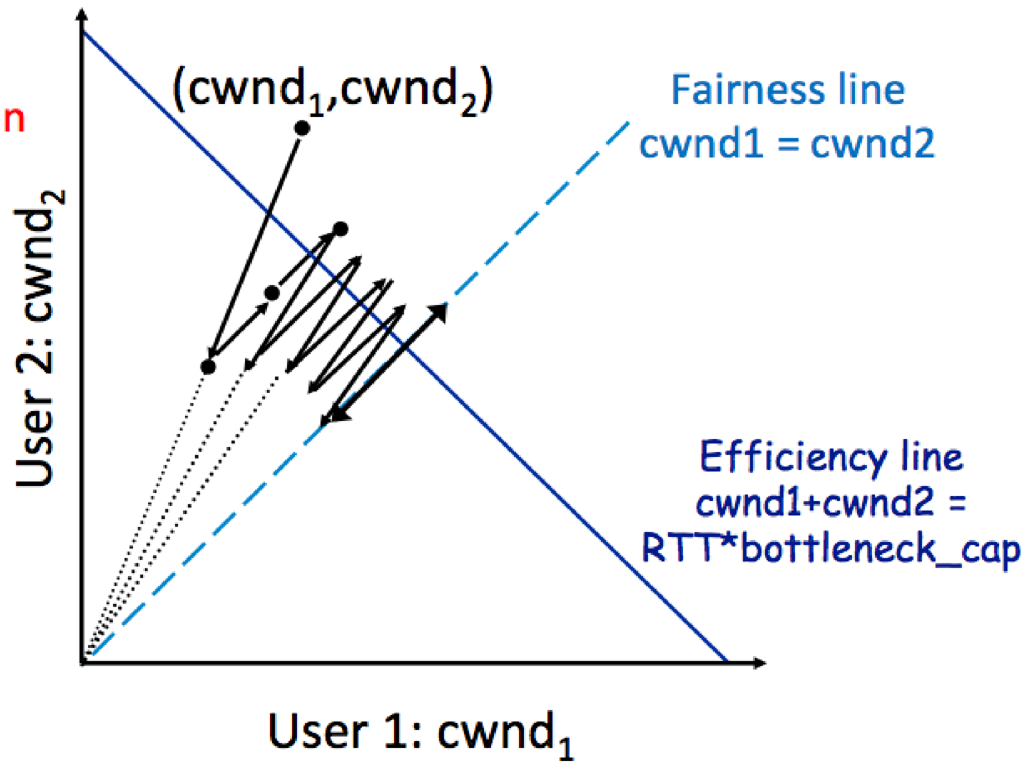


AIMD Leads to Efficiency and Fairness

Consider two users who have the same RTT

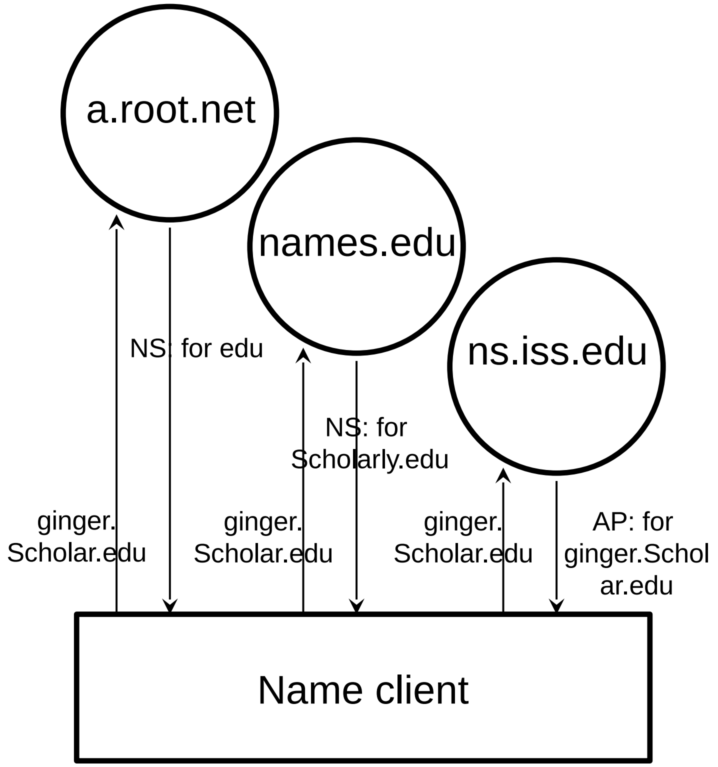
MD → move on lines through origin

AI → move on lines parallel to fairness line

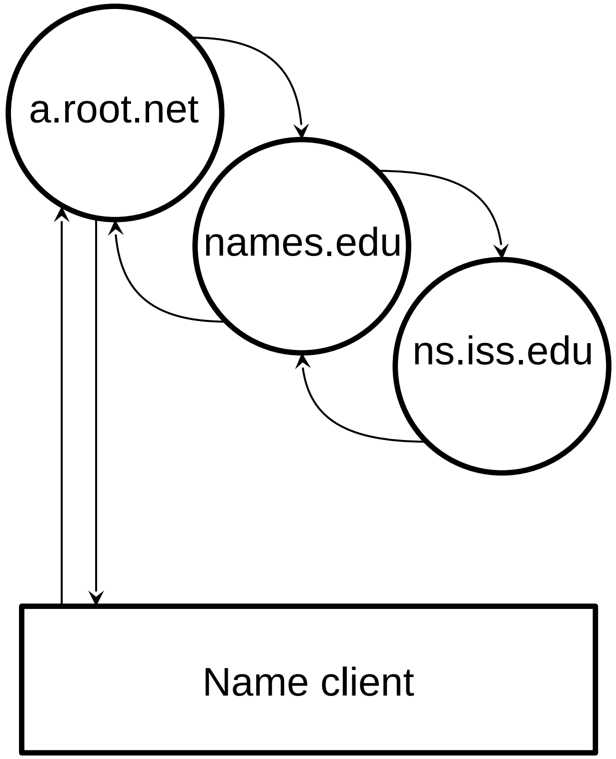


DNS Hierarchy (a partial view)

DNS Request Process

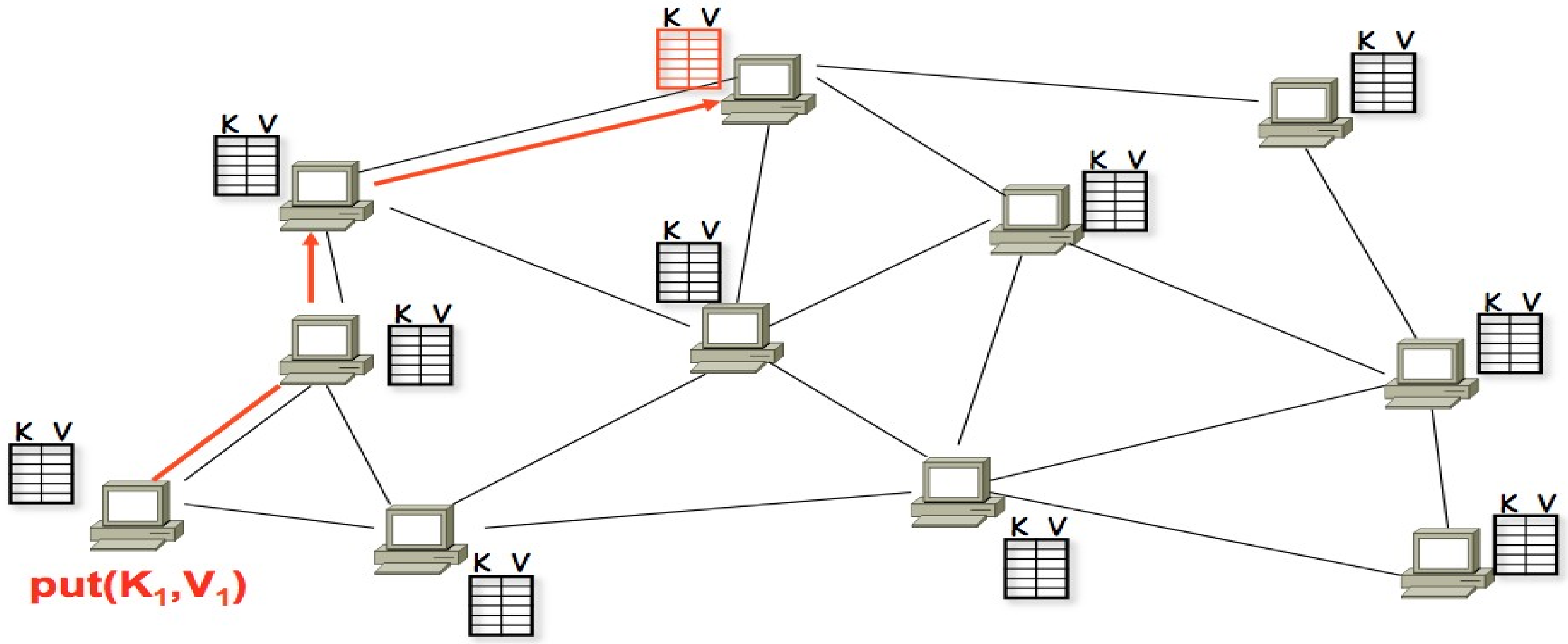


Non-Recursion

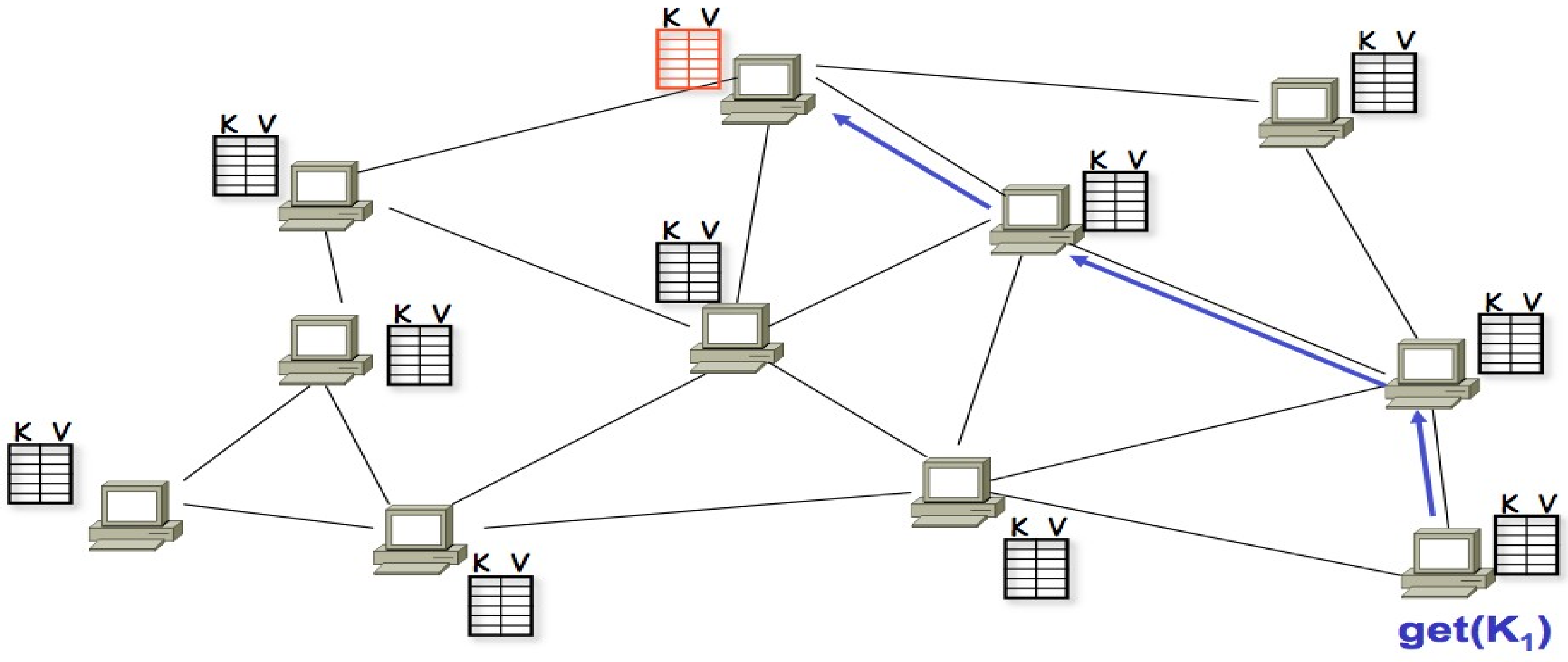


Recursion

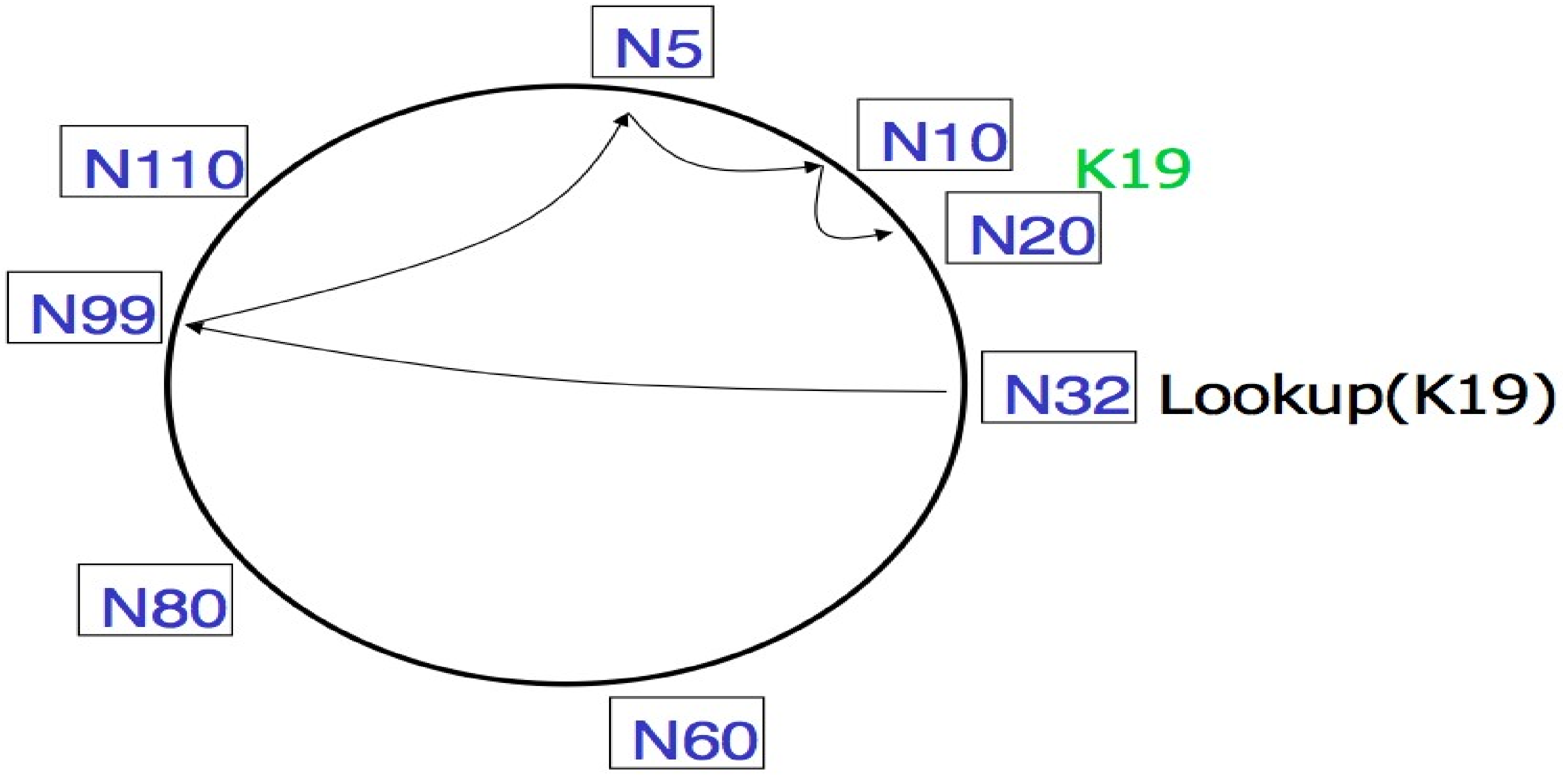
A DHT in Operation: put()



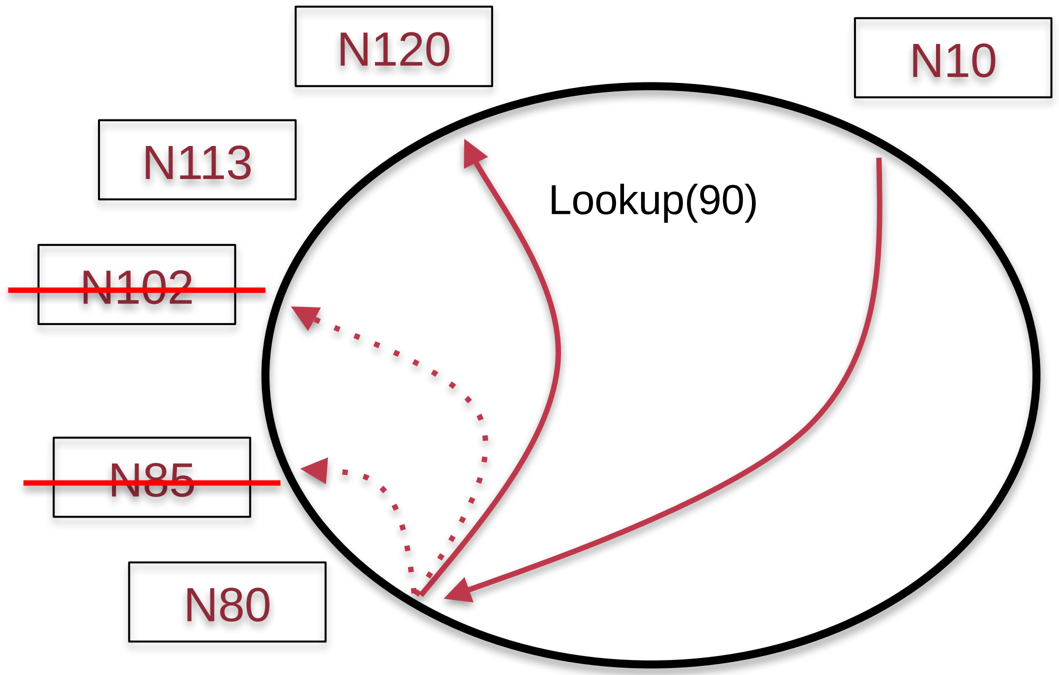
A DHT in Operation: get()



Lookups take O(*log(N)*) hops



Failures might cause incorrect lookup



N80 doesn't know correct successor, so incorrect lookup

Review: Raft and Distributed Computation

Review: Replication is everywhere for storage

For performance

- Higher throughput: replicas can serve concurrently
- Lower latency: cache is also a form of replication

For fault tolerance

- Maintain availability even if some replicas fail

Challenge: Replication Consistency

Optimistic Replication (e.g., eventual consistency)

- Tolerate inconsistency, and fix things up later
- Works well when out-of-sync replicas are acceptable

Pessimistic Replication (e.g., linearizability)

- Ensure strong consistency between replicas
- Needed when out-of-sync replicas can cause serious problems

Review: Replicated State Machines (RSM)

A general approach to making consistent replicas of a server:

- Start with the **same initial state** on each server
- Provide each replica with the **same input** operations, in **same order**
- Ensure all operations are **deterministic**
  - E.g., no randomness, no reading of current time, etc.

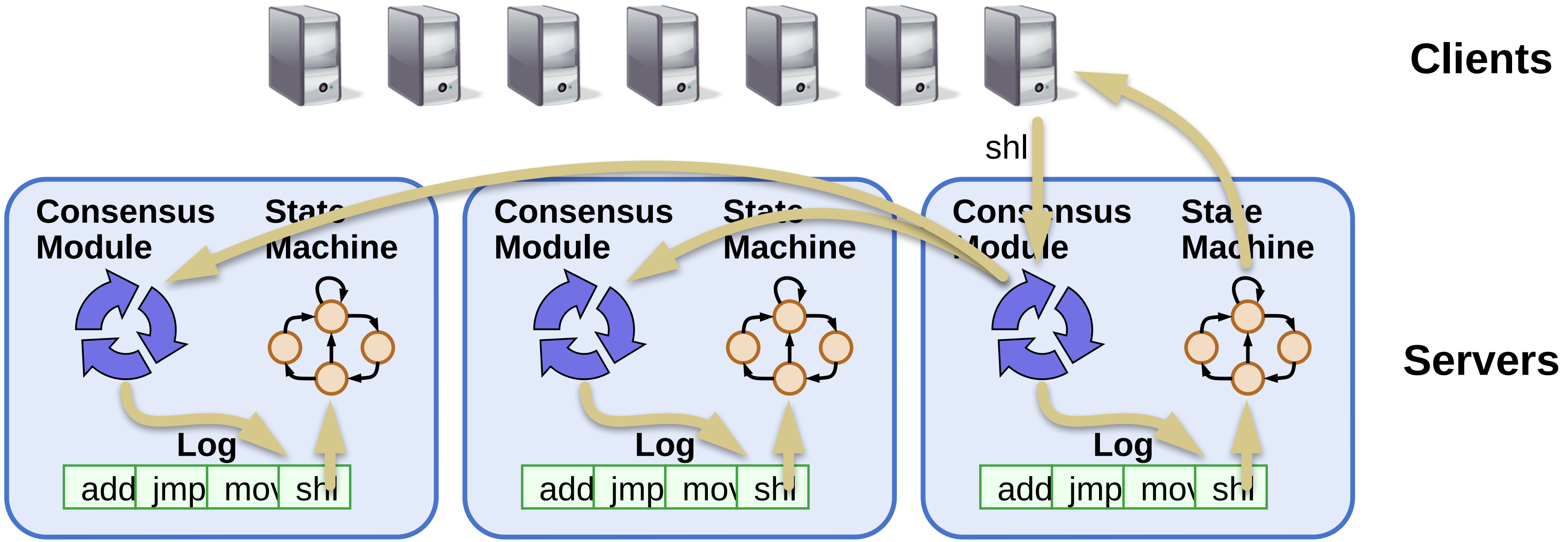
These rules ensure each server will end up in the same final state

Review: Inconsistency of Replicas

Problem: replicas can become inconsistent

- Issue: clients' requests to different servers can arrive in different order
- How do we ensure the servers remain consistent?
  - Unlike optimistic replication (e.g., eventual consistency), we cannot re-order events later, we must order it right now

Raft for RSM: replicated log



Replicated log => replicated state machine

– All servers execute same (deterministic) commands in same order

Consensus module ensures proper logs are the same!

Raft’s high-level approach: problem decomposition

1. **Leader election**

    Select one server as the leader

    Detect crashes, choose new leader

2. **Log replication (normal operation)**

    Leader accepts commands from clients, append to its log

    Leader replicates its log to other servers (overwrites inconsistencies)

3. **Safety**

    Keep logs consistent

    Only servers with up-to-date logs can become the leader

Raft server state switch: Heartbeats & Timeouts

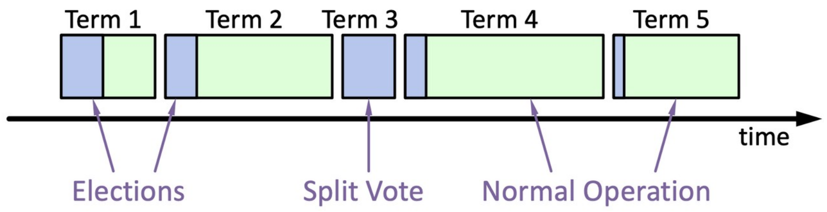
Servers start as followers (except one server is assigned as the leader)

- Followers expect to receive RPCs from leaders or candidates

Leaders must send heartbeats to maintain authority in a term

- If election timeout elapses with no RPCs:
  - Follower assumes leader has crashed
  - Follower starts **new election**
  - Timeouts typically 100-500ms

Review: Election Basics



1. Change its state to the candidate state
2. Increment current term
3. Vote for itself

   Send **`RequestVote RPC`**s to all other servers, retry until either

   ① Receive votes from majority of servers ➔ Become the leader!

   ② Receiver RPC from a valid leader ➔ Return to the follower

Question: what is the condition for the follower to vote?

The leader’s term >= self’s term & never vote before

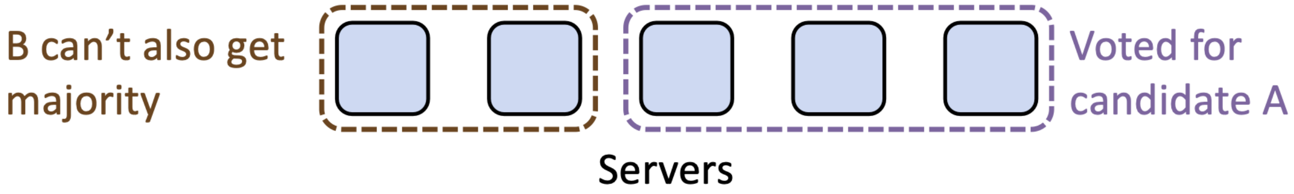
Election requirements: safety & liveness

At most one winner per term

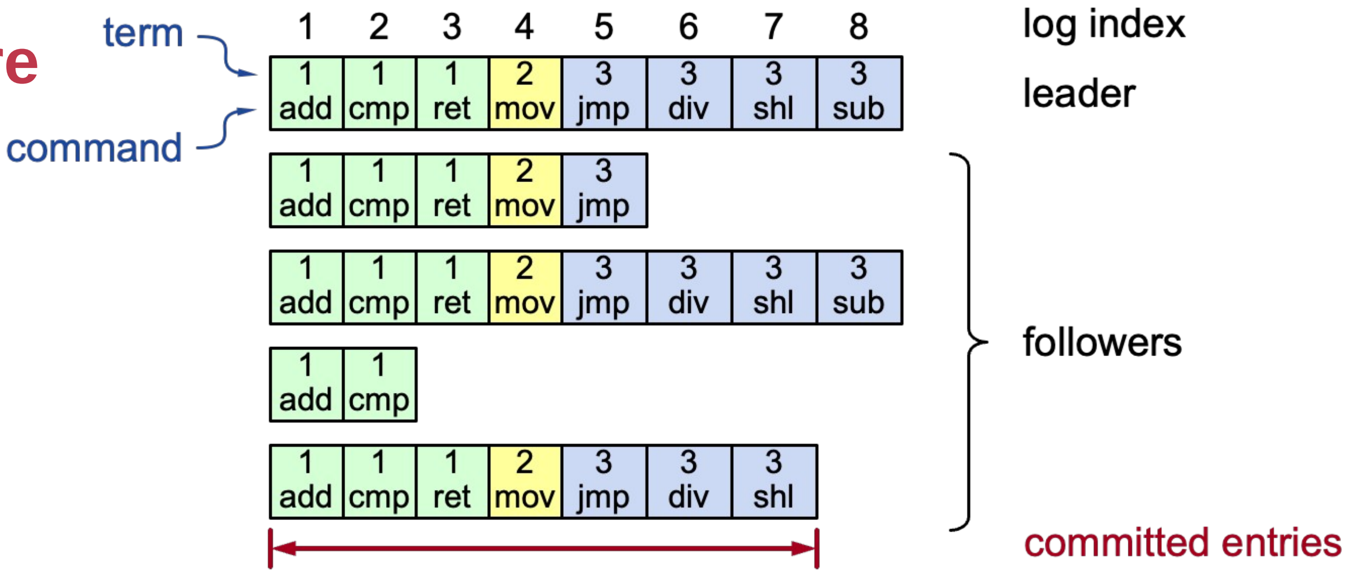
- How to achieve so? Each server only gives one vote per term (persist on disk)
- Majority ensures at least one leader (similar to phase #1 of Paxos)

Some candidate must eventually win

- If multiple candidates started, non will win



Log structure



term
command
1 2 3 4 5 6 7 8
log index
leader
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp | 3 div | 3 shl | 3 sub
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp | 3 div | 3 shl | 3 sub
1 add | 1 cmp
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp | 3 div | 3 shl
followers
committed entries

Log entry = index, term, command

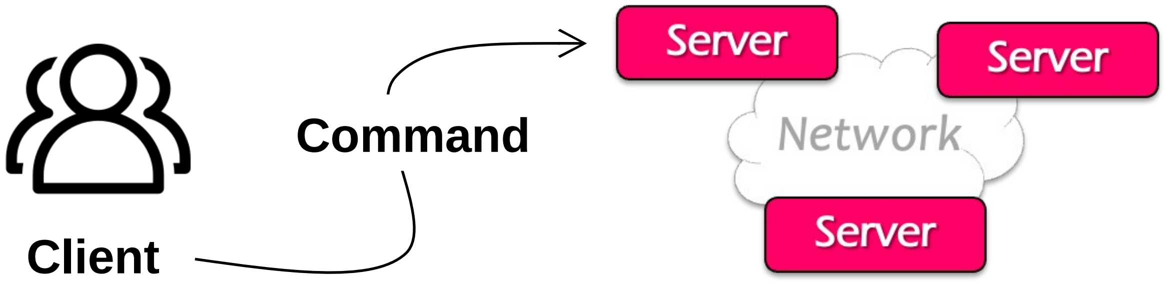
– Stored on the disk to tolerate failures

A log is committed if it can be safely applied to the state machine

– i.e., eventually stored on all the servers with the same value

Not all entries are committed (will talk about later)

Review: normal operations to update the log



① Send command to the leader

② Leader appends command to its log

③ Leader sends `AppendEntries` RPCs to followers

④ Once a new entry (of log) **committed**:

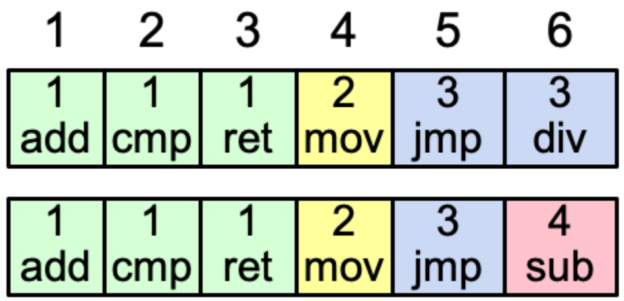
Leader passes command to its state machine, returns results to client

Notifies followers of committed entries, Follower pass committed commands to their state machines

Consistency of the raft’s log

High level of coherency between logs maintained by the raft:

– If log entries on different servers have the same index & term
  - They store the same command
  - The logs are identical in all preceding entries


1 2 3 4 5 6
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp | 3 div
1 add | 1 cmp | 1 ret | 2 mov | 3 jmp | 4 sub

If a given entry is committed, all preceding entries are also committed

– Note that not all log entries are committed

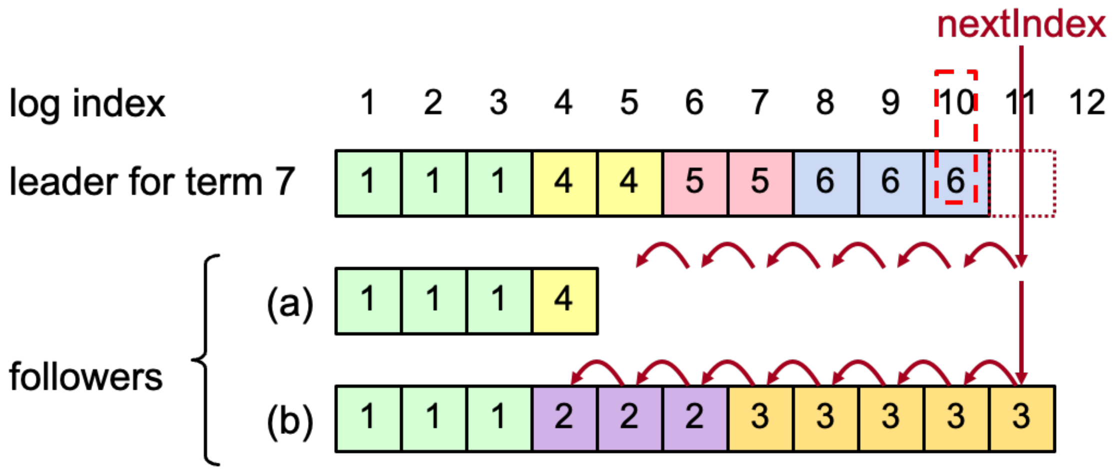
AppendEntries consistency checks + repair

Each RPC argument contains

– Append index, term, **term of entry preceding new ones**

Follower checks whether it has the matching entry

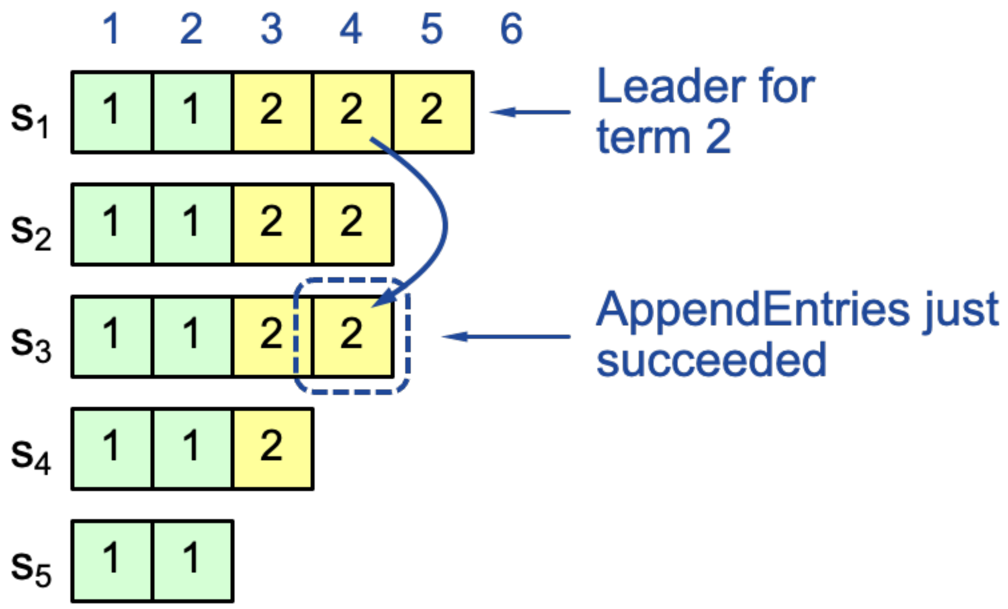
– Otherwise, it rejects the request



Raft’s intuitive commit rule: replicating on a majority

If the log is replicated on a majority of servers, it can be replicated on the state machine

– Not always true on raft so far



Safety requirement of the commit entry

Once a log entry has been applied to a state machine, no other state machine must apply a different value for that log entry

Raft safety property:

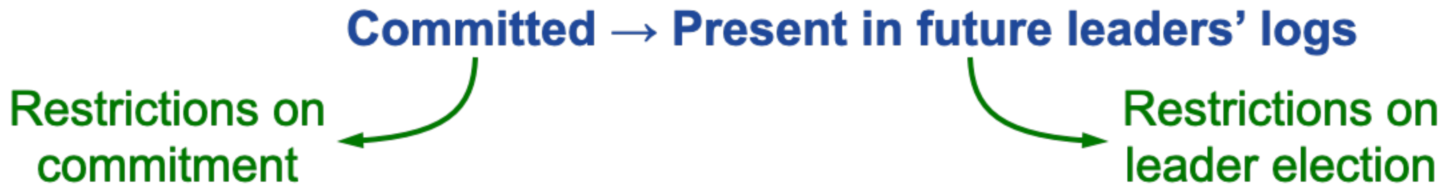
– If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders (no overwritten)

This guarantees the safety requirement

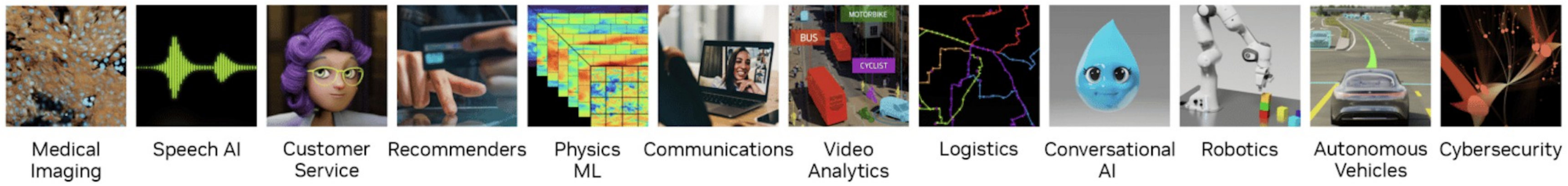
– Leaders never overwrite entries in their logs

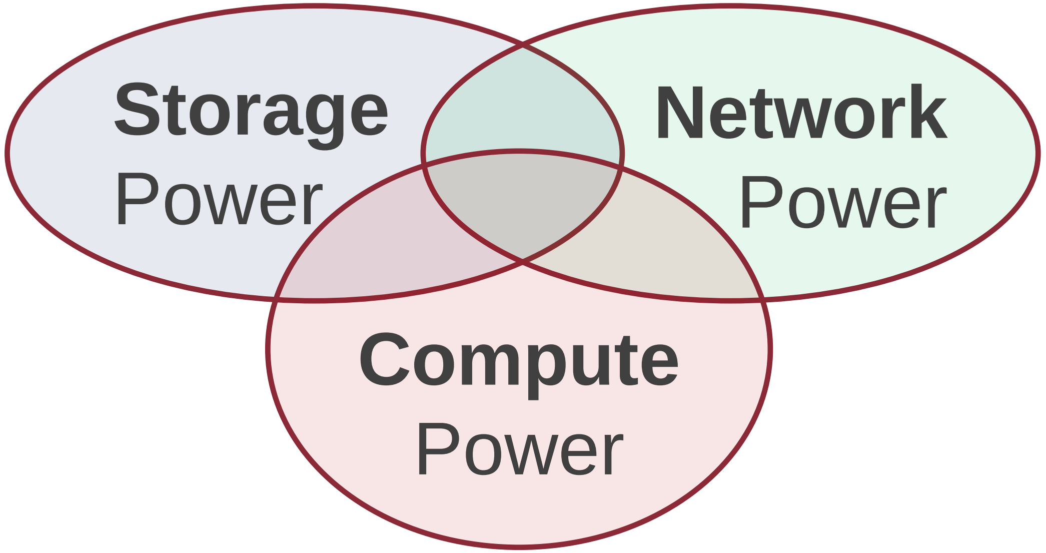
– Only entries in the leader's log can be committed

– Entries must be committed before applying to state machine


Committed → Present in future leaders' logs
Restrictions on commitment
Restrictions on leader election

Systems: storage, compute and network





Review: (Approximate) computation for NN like ChatGPT

Cost of training ~=

Known fact: GPT-4 has 1.76 trillion parameters [1]

- This is 1,760,000,000,000 parameters
- So, this is 10,560,000,000,000 calculations for a single input of **a single iteration**!!

What are the computation capabilities of nowadays devices (e.g., A100)?

- 19.5 TFLOPS = 19,500,000,000,000 (FP32) float point per second
- Basically, it needs 30 seconds for an A100 GPU to finish an iteration **in the optimal case**

We need a powerful computation device for the AI!

[1] https://the-decoder.com/gpt-4-has-a-trillion-parameters/

Review: scaling a single-device with parallelism

General methods

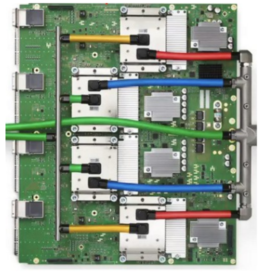
- Single core+: pipeline + super scalar with instruction level parallelism (ILP)
- Single core++: added SIMD support
- Single core+++: domain-specific accelerators (e.g., matrix accelerators)
- Multiple core: a single core (single core, single core+, single core++, single core+++) can be glued together !

Question

- What is the trade-off?

Review: Devices available for computation

Programmability ←



Intel i5-9600K, single core: 6.3GFLOPS Multiple cores: 37.7GFLOPS

Mate60 GPU 2.3 TFLOPs

Apple M2 GPU 3.6 TFLOPs

NVIDIA A100 GPU 19.5 TFLOPs

Google TPUv4 275 TFLOPS

→ **Performance**

Simply add computation is insufficient!

Accessing memory (or other storage) & Roofline model

Memory stalls (Communication stalls)

A processor "stalls" when it cannot run the next instruction in an instruction stream because of a dependency on a previous instruction.

Accessing memory is a major source of stalls

```
ld r0 mem[r2]
ld r1 mem[r3]
add r0, r0, r1
```

Dependency: cannot execute 'add' instruction until data at mem[r2] and mem[r3] have been loaded from memory

Memory access times ~ 100's of cycles

– Memory "access time" is a measure of latency

Reducing memory stalls is a large topic, out of the scope of this lecture

Terminology

Memory (storage) latency

- The amount of time for a memory request (e.g., load, store) from a processor to be serviced by the memory system
- E.g., 100 cycles, 100 nsecs

Memory (storage) bandwidth

- The rate at which the memory system can provide data to a processor
- E.g., 20Gbps

Time to load the data = Latency + Payload / bandwidth

Review: the roofline model

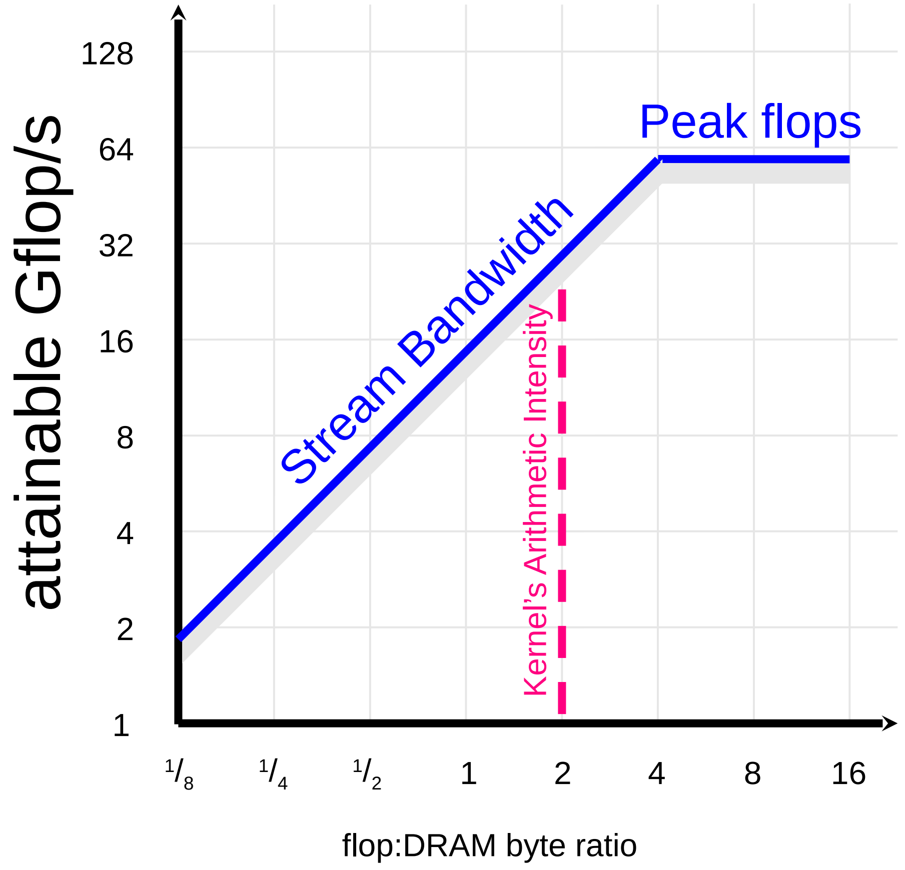
A computation task should look at capabilities other than the computation

Given an application

- If we know how many FLOPs performed per-memory read, we can see whether it is computation bound or memory bound

Benefit

- Give hints on the optimization directions



Review: Why distributed computing?

Large computation requirements

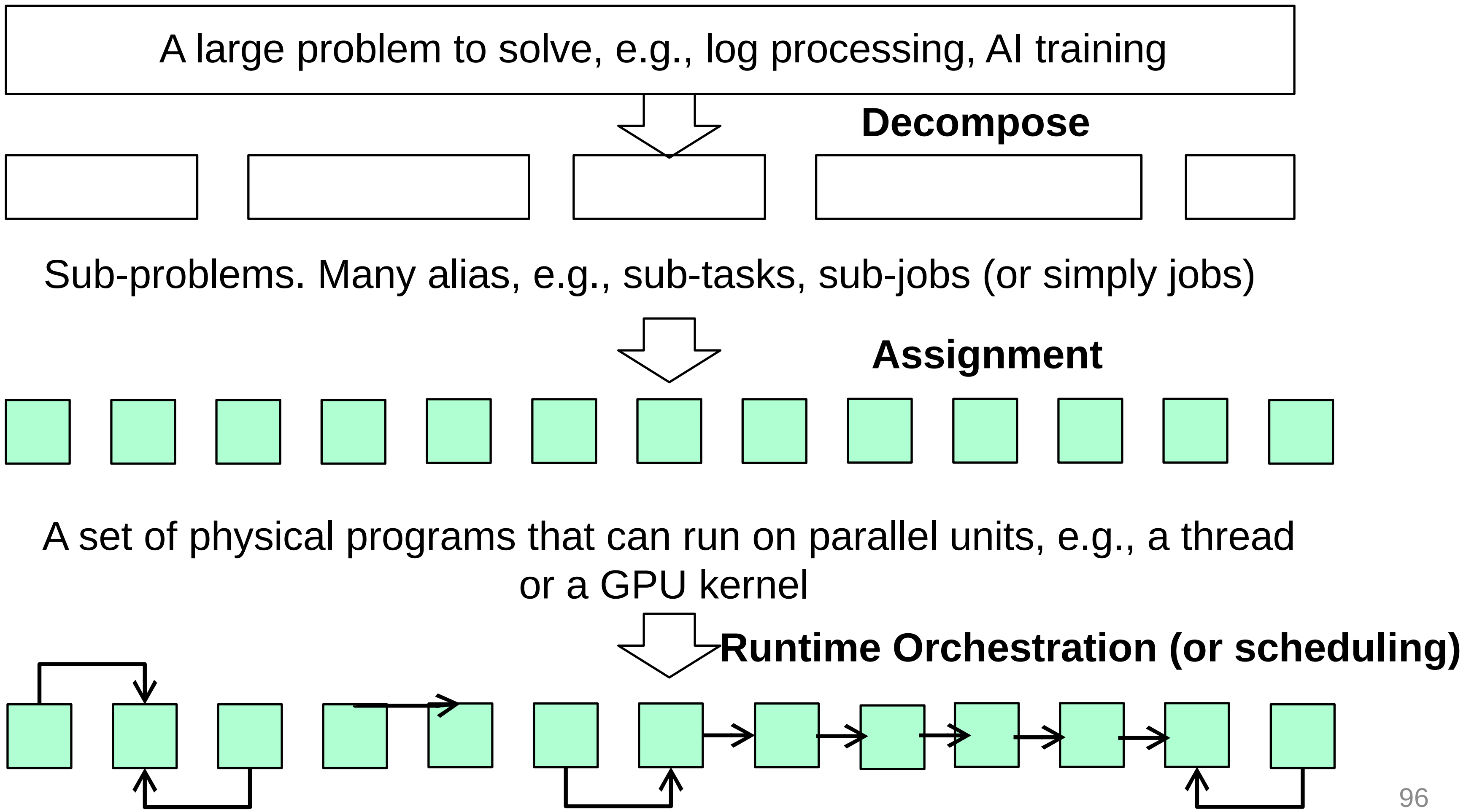
- E.g., training AI models, huge floating points required for each iteration

  = 6 * #parameters * Batch size
- The trend continues

Single device FLOPS (floating points per second) is insufficient

- 2X speed improvements per-year
- Training computation required: 240+ X required per-year

Logical process for doing a computation distributedly



A large problem to solve, e.g., log processing, AI training

Decompose

Sub-problems. Many alias, e.g., sub-tasks, sub-jobs (or simply jobs)

Assignment

A set of physical programs that can run on parallel units, e.g., a thread or a GPU kernel

Runtime Orchestration (or scheduling)

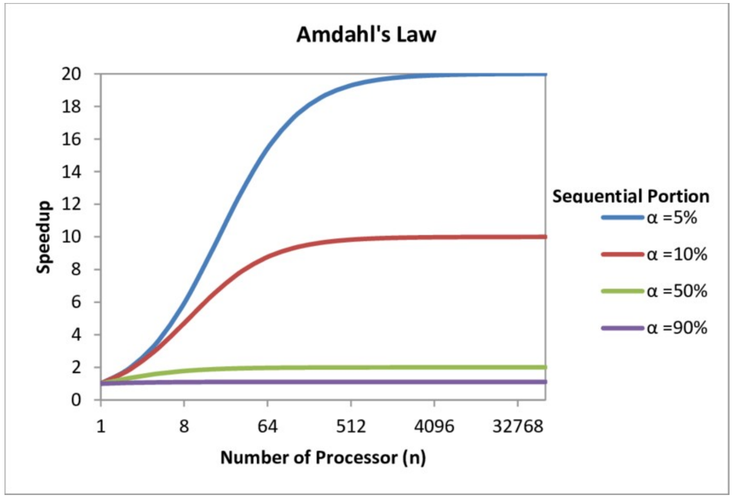
Review: Amdahl’s Law for decomposing the tasks

Let **= the fraction** **of total work that is** **inherently sequential**

Max speedup on M machines given by:

Speedup

Takeaway: we should reduce sequential part during distributed computation



Source: https://www.researchgate.net/figure/Graph-demonstrating-Amdahls-Law_fig2_315696585

More implementation details to consider

1. Sending **data** to/from nodes
2. **Coordinating** among nodes
3. Recovering from node **failure**
4. Optimizing for **locality**
5. Partition data to to enable more **parallelism**

Common to many jobs! E.g., batch processing, graph processing, machine learning, etc.

Review: we build a framework to hide the above tasks

Goal

- Reduce programmer’s effort in solving the above challenges

Challenges

- What are the abstraction? Thread & RPC is insufficient
- More general, harder to provide the above property
- E.g., if a thread fails, how can we know its progress?
  - It’s a very hard problem, we will come back to it later

Existing abstractions: MapReduce & Computation graph (DAG)

MapReduce vs. DAG

1. Sending **data** to/from nodes (both done)
2. **Coordinating** among nodes (both done)
3. Recovering from node **failure** (both done)
4. Optimizing for **locality** (both done)
5. Partition data to to enable more **parallelism** ?
   1. MapReduce: jobs are naturally parallelized w/ its abstraction
   2. But in DAG, the level of parallelism depends on the structure of DAG!

We need some domain-specific knowledge to help partition the graph node for more parallelism in DAG

Case study: distributed training

Review: Ideal Metric of Success for Efficient Training

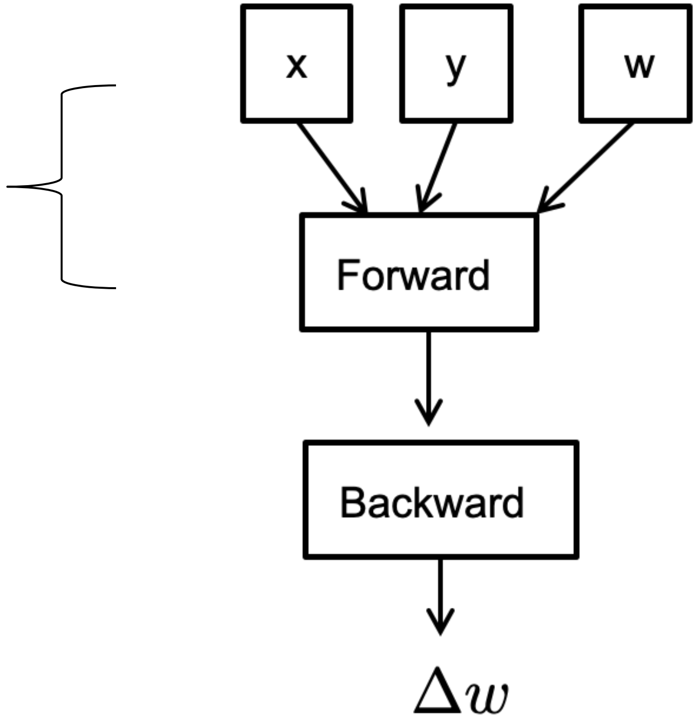
$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

Convergence
Machine Learning
Property

Throughput
System
Property

Review: Pseudocode for Stochastic Gradient Descent

```
// execute on a master
for iter in num_iters:
                                                iter

          iter+1 iter
```



Note: expressed as DAG

Review: Parallelization Opportunities

Data Parallelism: *Distribute the processing of data to multiple PEs.*

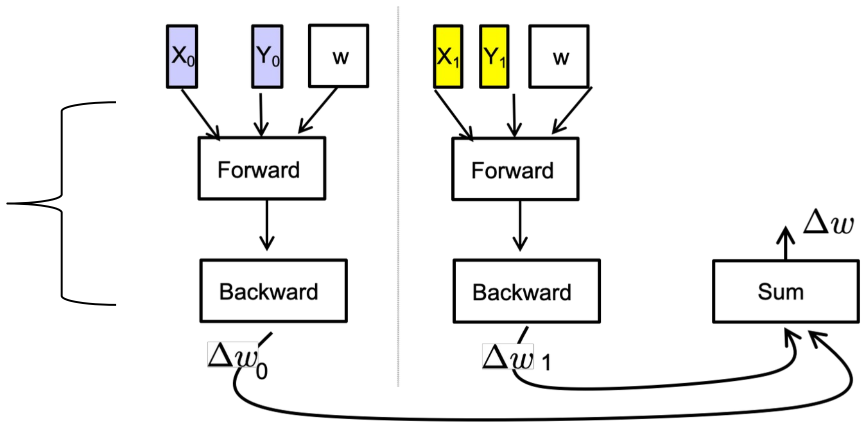
Model Parallelism: *Break the model and distribute processing of every layer to multiple PEs*

For either approach it is also possible to use ***synchronous*** *or* ***asynchronous*** *updates*

Review: Pseudocode for (iterative) data parallel

```
// execute on a master
for iter in num_iters:

          iter+1 iter
```

iter



The master will do the coordination

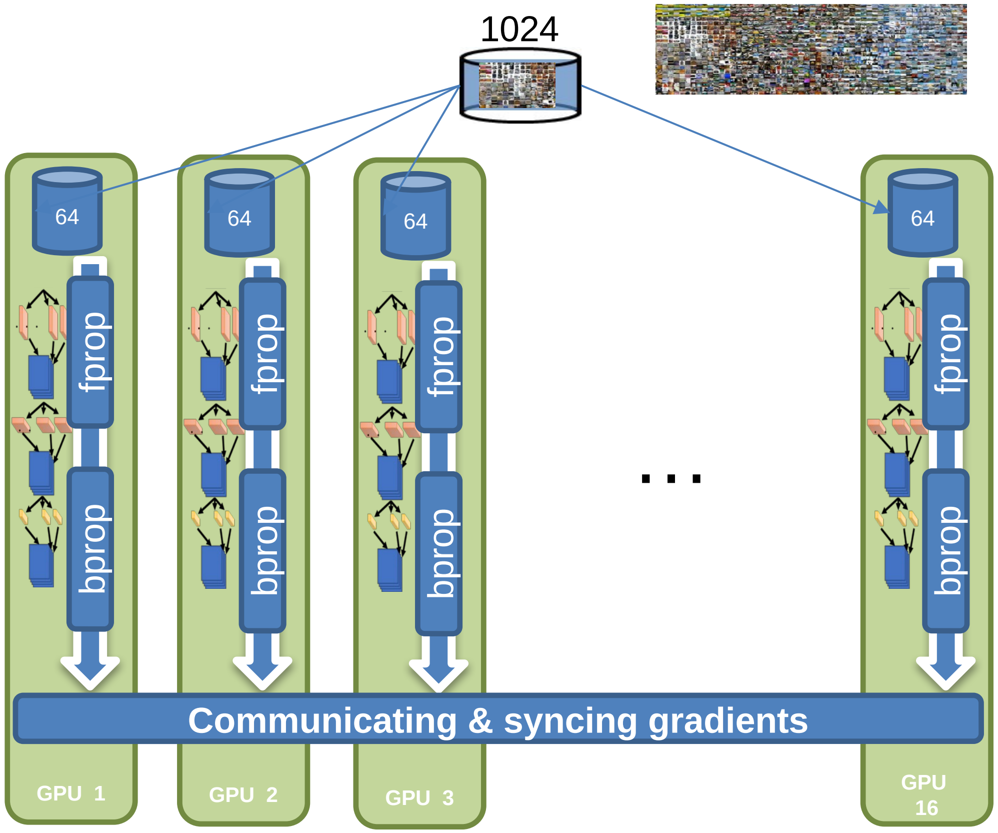
– Similar to the Job manager in Dryad

Synchronous Data Parallelism w/ BSP

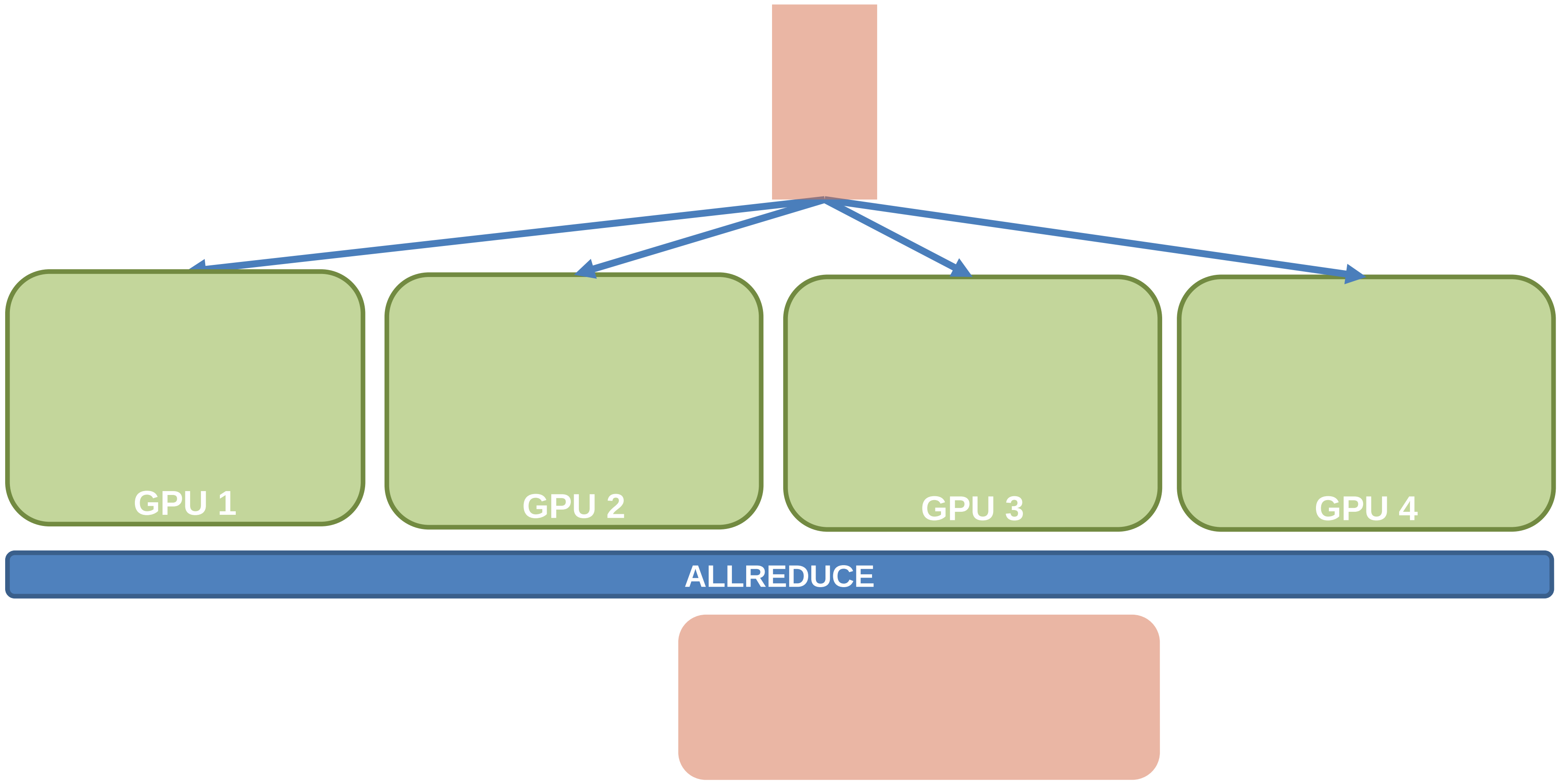
Store the entire model (W) on each processor

- Then distribute the batch evenly across each processor
- E.g., 64 images per GPU

Question: how to sync between iterations?



Key operator: allreduce



Allreduce: put it all together

Trade-off between rounds vs. fan-in performance

	Round	Peak node bandwidth	Per-node fan-in
Parameter server (PS)	O(1)	O(N * P)	O(P)
Decentralized Allreduce	O(P)	O(N)	O(1)
Ring allreduce	O(2 * P)	O(N/P)	O(1)

Drawback of data parallelism: replicated model

Data parallelism assumes the model (parameters) are replicated on all the processes

- Such that they can do the forward & backward pass dependently
- What if the model cannot fit onto the device?

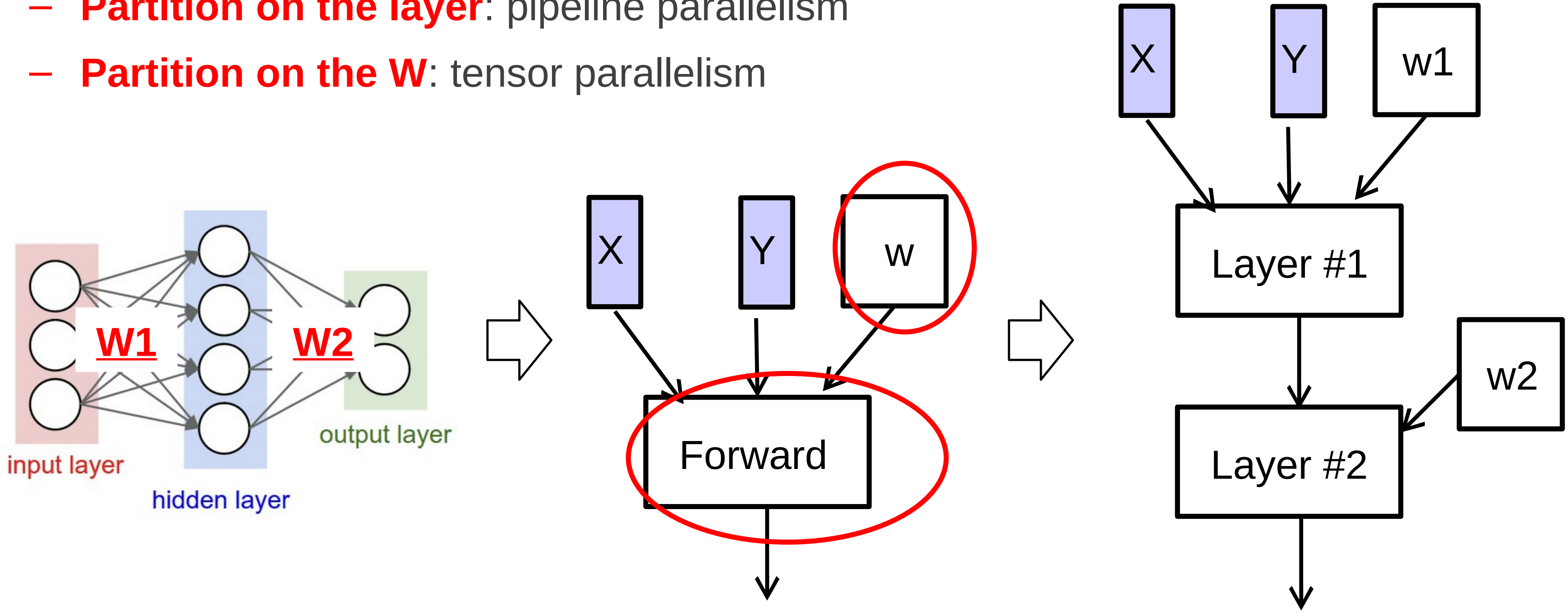
Current trends: models are becoming larger and larger

- No country (or company) can tolerate the risk of falling behind in the AI race

Model parallelism

Partition the parameters of a model, typically done in two ways:

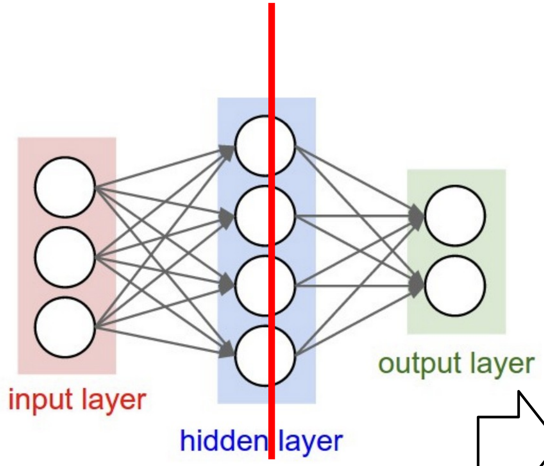
- **Partition on the layer**: pipeline parallelism
- **Partition on the W**: tensor parallelism



Pipeline parallelism

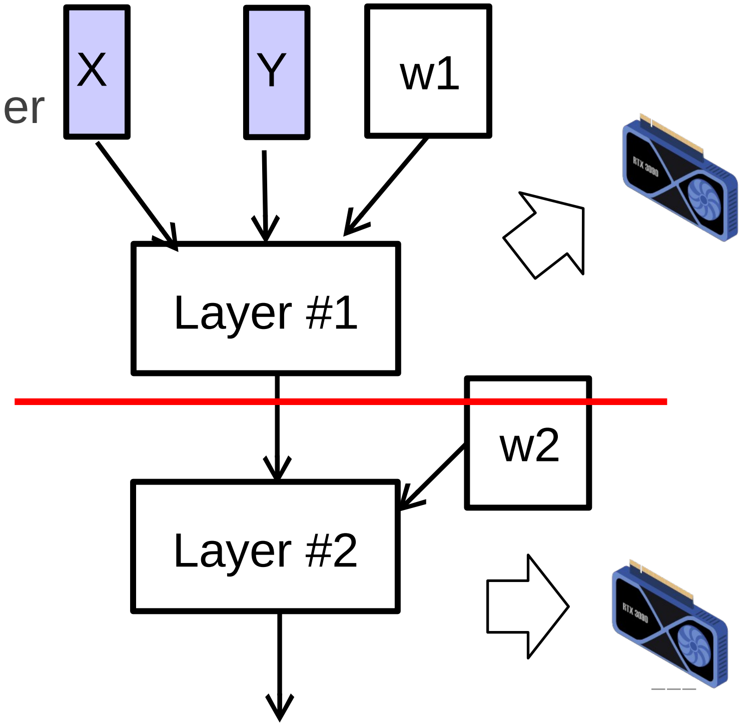
Partition the computation layer-by-layer, each partition is deployed on a device

- Note that different layers can be grouped together
- The strategy is out of the scope of this course

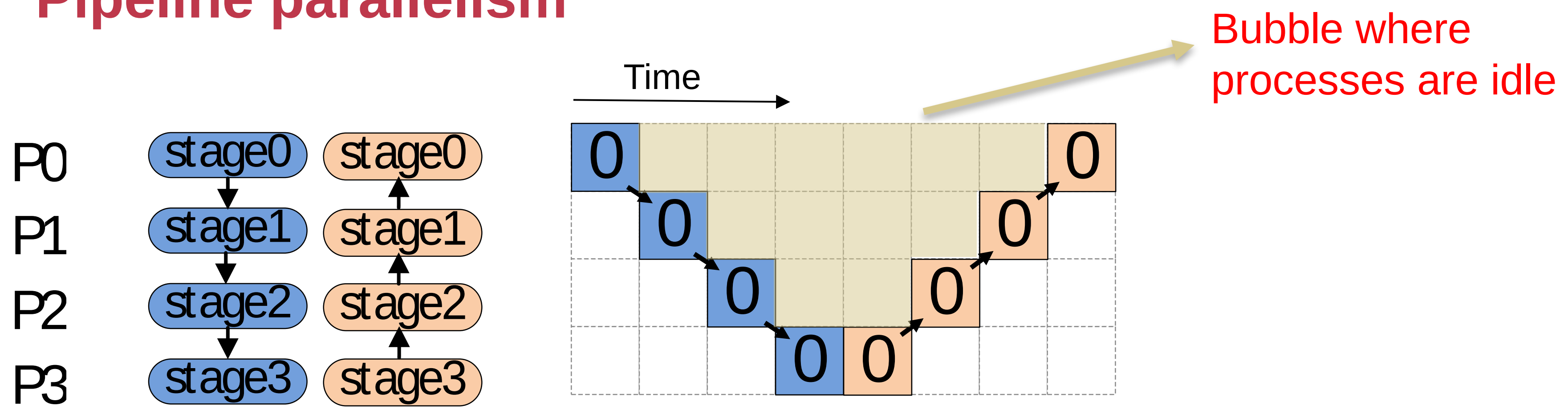


The model

The partitioned computation graph



Pipeline parallelism



Problem: no parallelism there!

	Bubble
X X	Forward and backward passes of ***model replica*0** for micro-batch **x**

M_θ	Memory consumption for the weights
M_a	Memory consumption for the activations

How to reduce bubble?

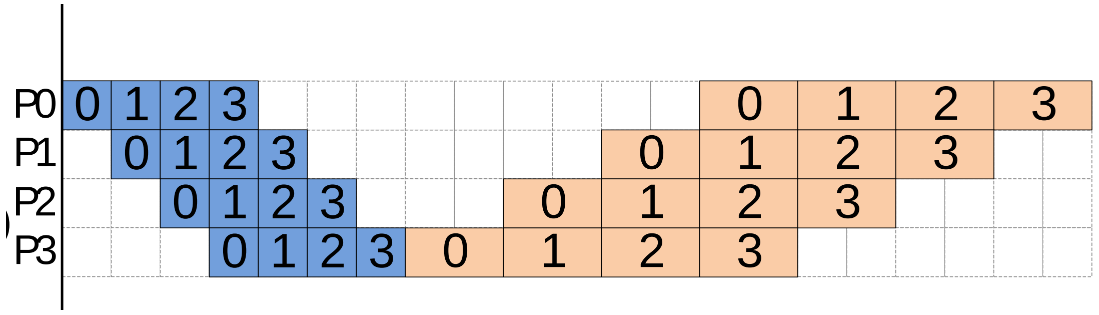
Technique: micro-batching

Reducing the bubble size by breaking the batch size into smaller pieces to reduce the idle time of the processes

- Reduces bubble size in an easy way to implement
- The batch size = partition size

Question: to what extent can micro-batching improves performance?

- Depends on the bubble size



Pipeline parallelism & the number of micro-batch

Question: what is the bubble size ?

- p – 1 (p == #devices)

Question: what is the overhead?

- : time of a forward of a micro-batch (m)
- : time of a backward of a micro-batch (m)
- Ideal process time:
- Wasted time of bubble:
- Bubble time fraction:

Tensor parallelism

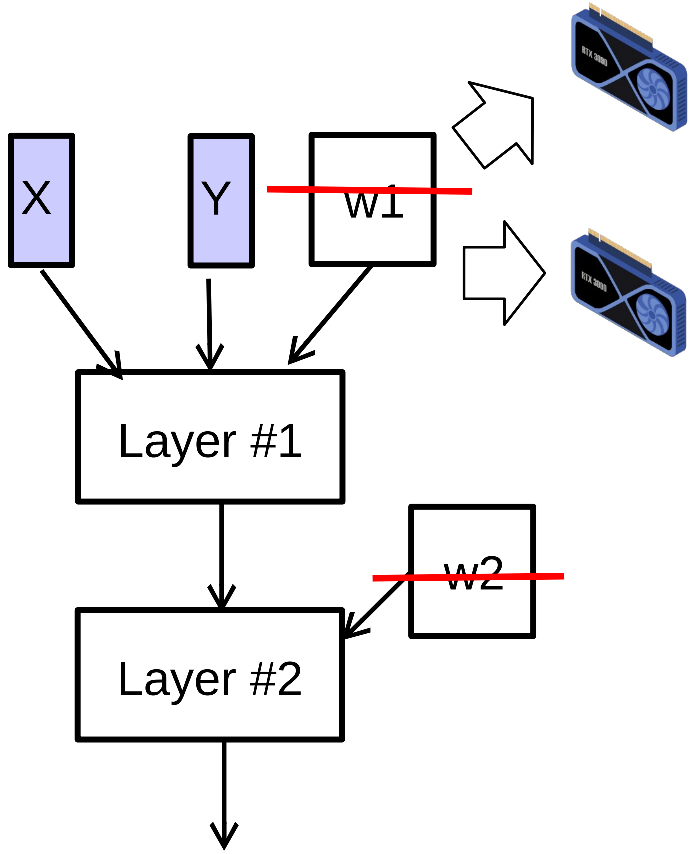
Partition the parameters of a layer

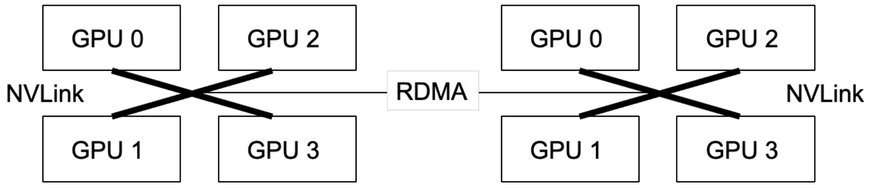
– Each partition is deployed on a separate GPU

Pros

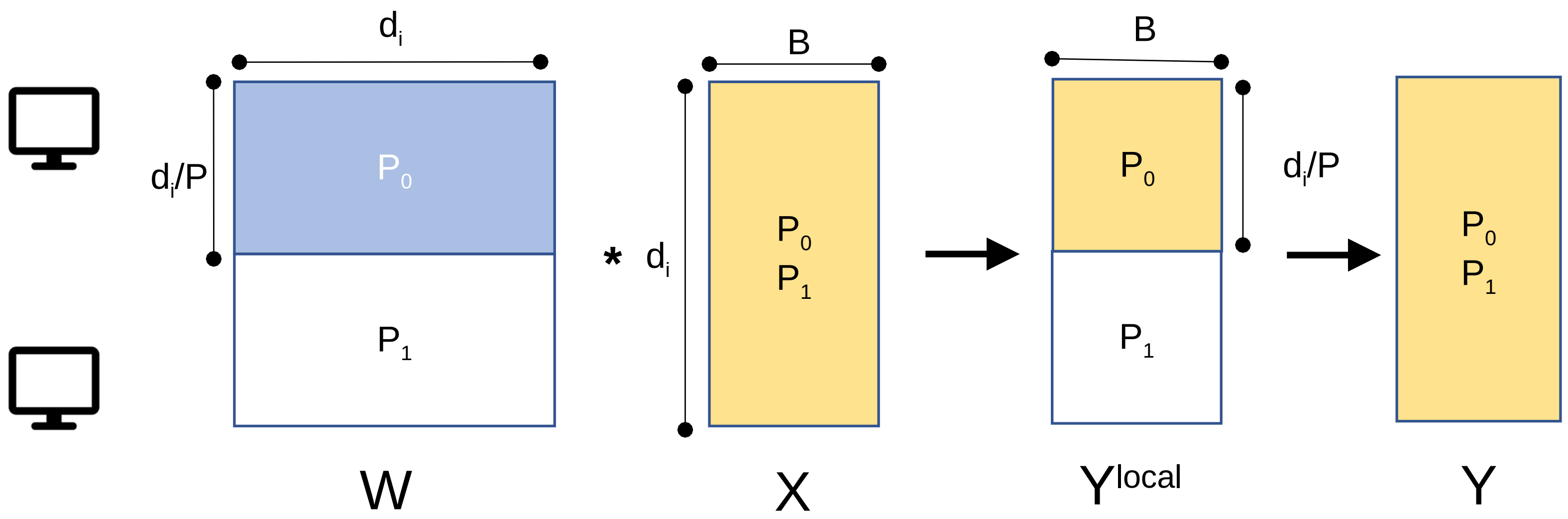
– Support pipeline parallelism with a large layer

– Fit the hardware architecture of modern servers (or we can say that modern servers are built for tensor parallelism ⏪)



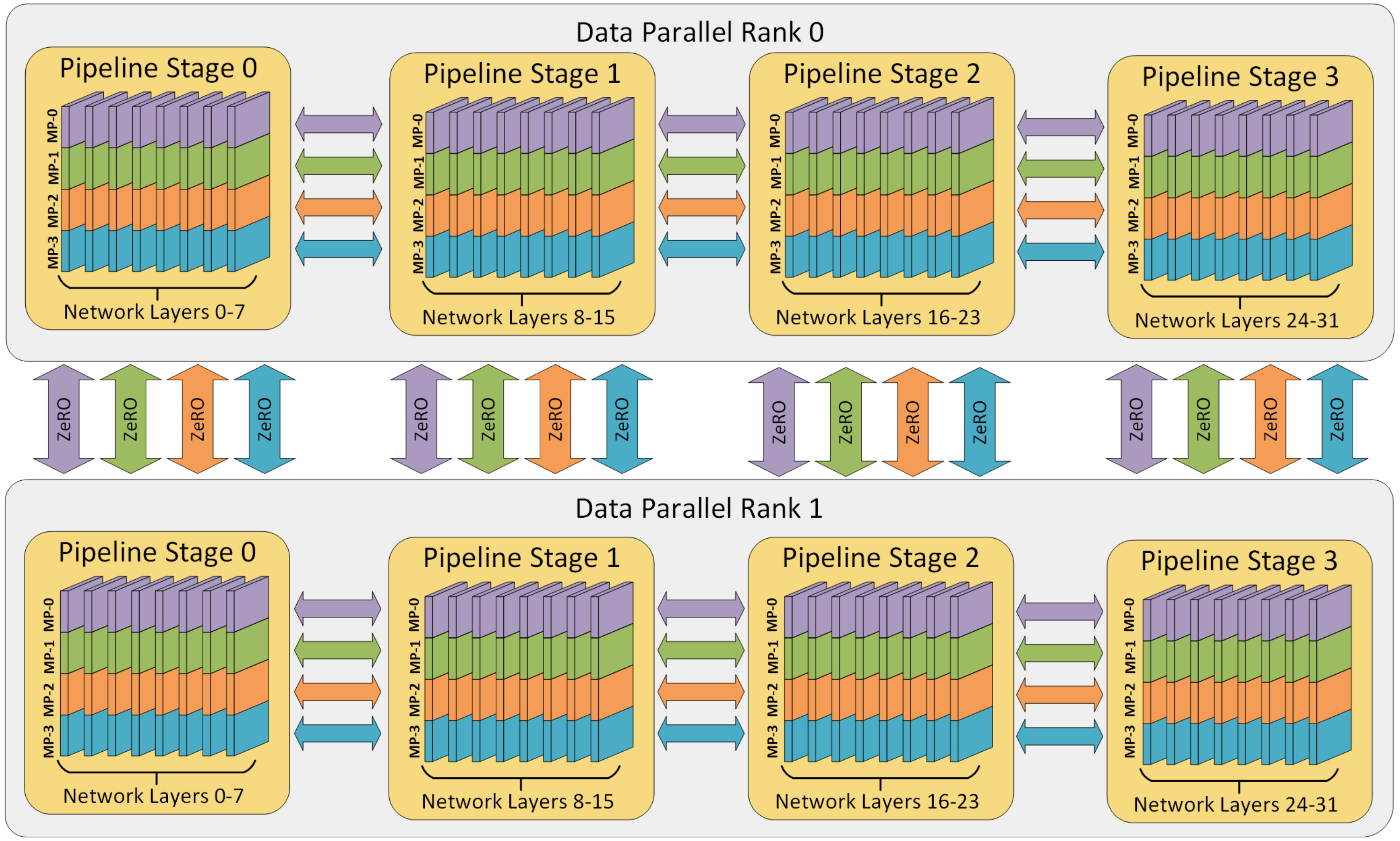


Model Parallelism: Forward Pass



- Requires an all gather communication so that all processes get each others activation data
- Same cost as all reduce without the 2x factor

Put it altogether: parallelize distributed training


Data Parallel Rank 0
Pipeline Stage 0
MP-3 MP-2 MP-1 MP-0
Network Layers 0-7
Pipeline Stage 1
Network Layers 8-15
Pipeline Stage 2
Network Layers 16-23
Pipeline Stage 3
Network Layers 24-31
ZeRO
Data Parallel Rank 1
Pipeline Stage 0
Network Layers 0-7
Pipeline Stage 1
Network Layers 8-15
Pipeline Stage 2
Network Layers 16-23
Pipeline Stage 3
Network Layers 24-31