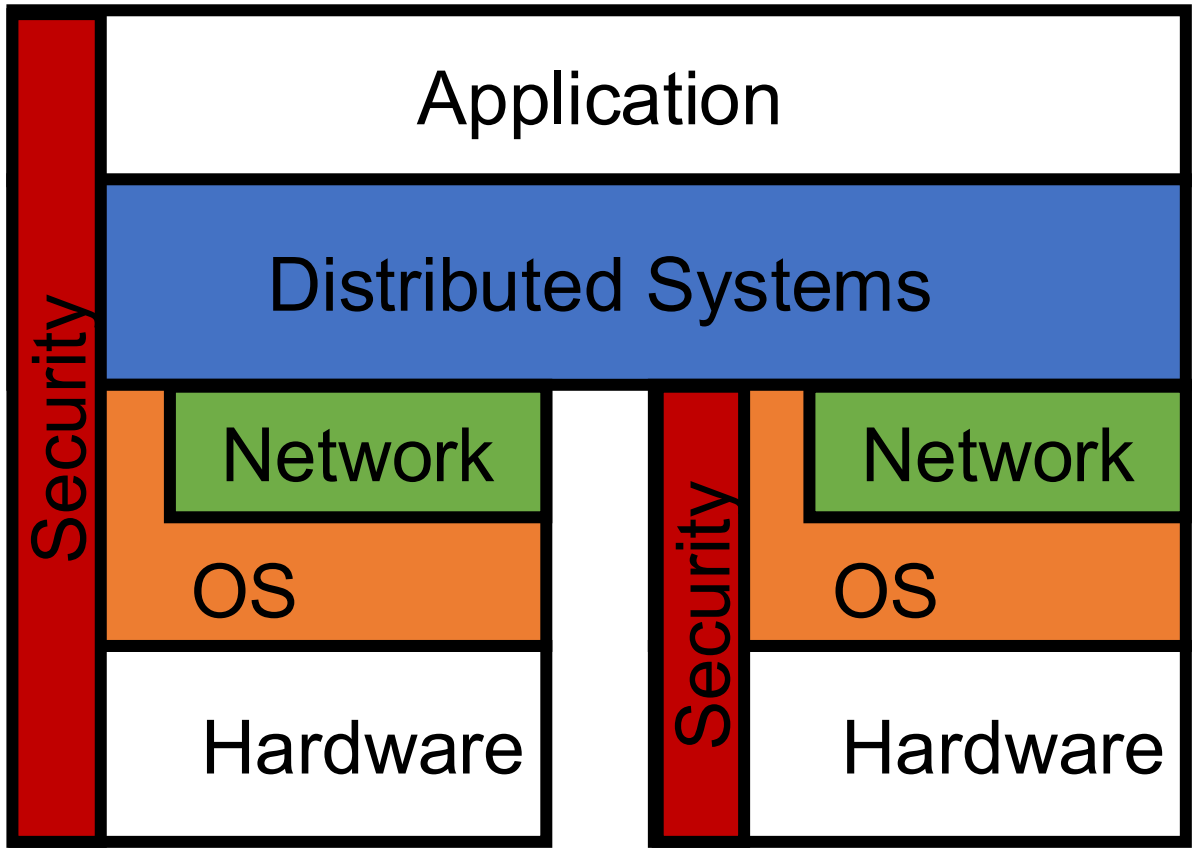
Tying It All Together

COS 316: Principles of Computer System Design

Lecture 20

Wyatt Lloyd

Types of Systems We Covered



- Distributed Systems
- Networking
- Operating Systems
- Security

High Level Topics Covered

- Systems
- Naming
- Caching
- Layering
- Concurrency
- Access Control

A “Simple” Example – Streaming Video

1. Record video on phone
2. Video sent over Internet to service
3. Web server receives video segments
4. Web server forwards segments to distributed file system
5. Web server initiates video processing
6. Video processing produces streamable versions
7. Video now streamable – shared w/ other users
8. User’s app fetches file with metadata about video segments
9. User’s app runs ABR algorithm to download video segments via CDN

1) Record video on phone

- Does app have access to video device?
- Interface to video device via OS
- Interface to storage via OS

2) Video sent over Internet to the service

- Host name -> IP address (e.g., youtube.com -> 172.217.10.14)
  - Naming!
- Global IP routing to 172.217.10.14
  - Layering!
- Sent over a TCP connection to a remote web server
  - Send whole video, error detection, congestion control, flow control
- Applications use socket interface
  - Assignment 1!

3) Web server receives video segments

- Use request routing logic to run handler for video segments
  - Assignment 2

- Is user authorized to create new videos?

4) Web server forwards segments to distributed file system

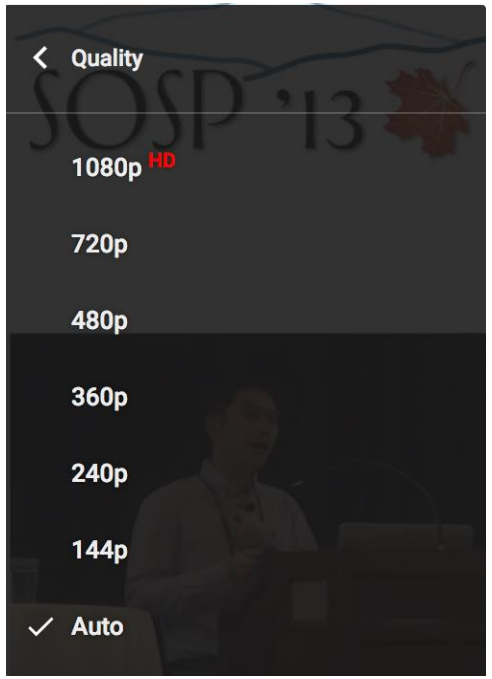
- Durability of video segments

- Distributed file system – looks (kinda) like a unix file system
  - On different machines, accessed over network, running on top of local unix file system

- Aside: video segment metadata
  - Bug: eventual consistency vs. linearizability

5) Web server initiates video processing

- Validate video, fix audio alignment, …
- Produce many different bitrates
- Compress video segments
- Generate thumbnails
- …



- Processing done by a distributed system

6) Video processing in action

- Many machines processing different segments of video in parallel
  - Concurrency!
- Durably store resulting video segments
- Following grey slides from Qi Huang's SOSP '17 talk on SVE

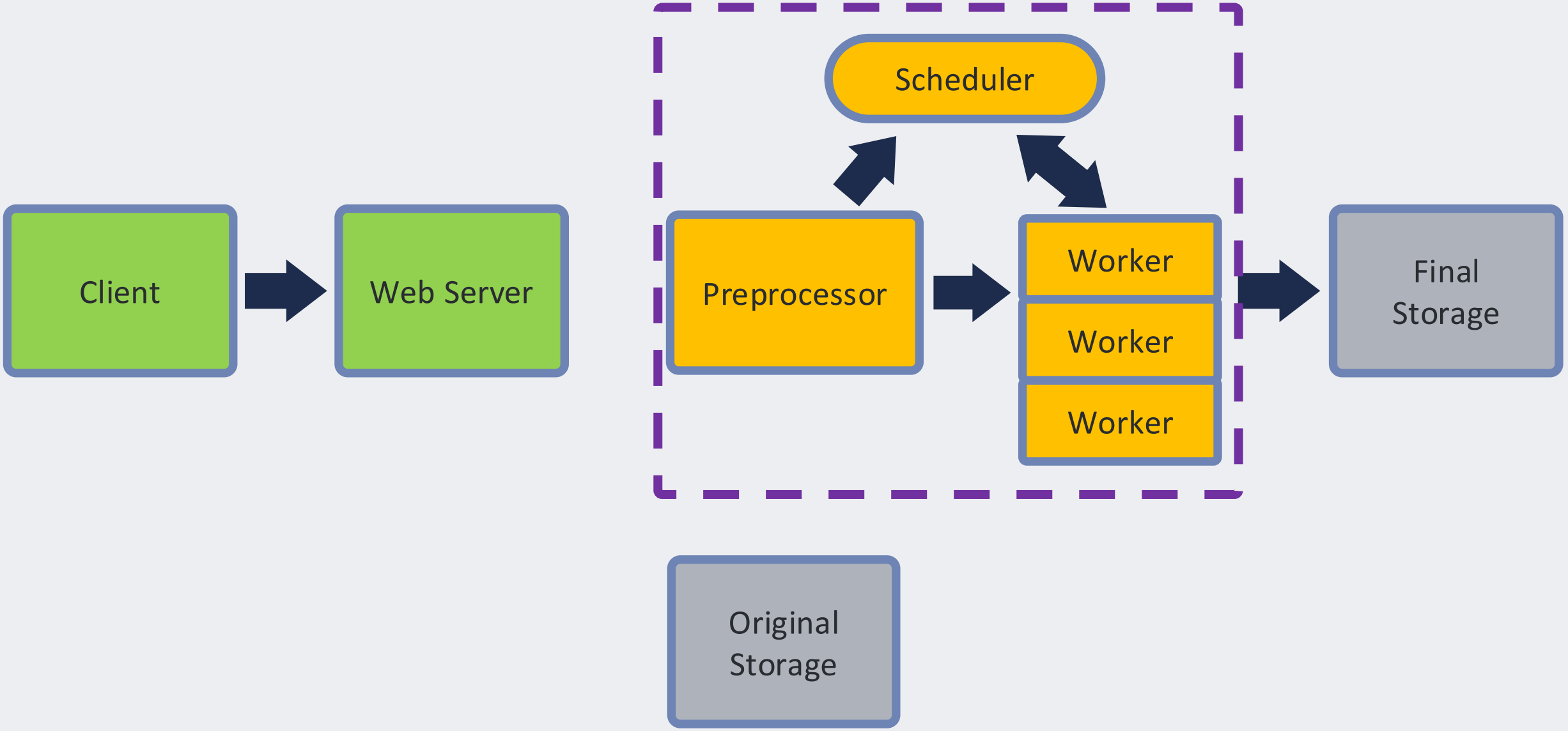
Speedy: harness parallelism

Users can share videos quickly

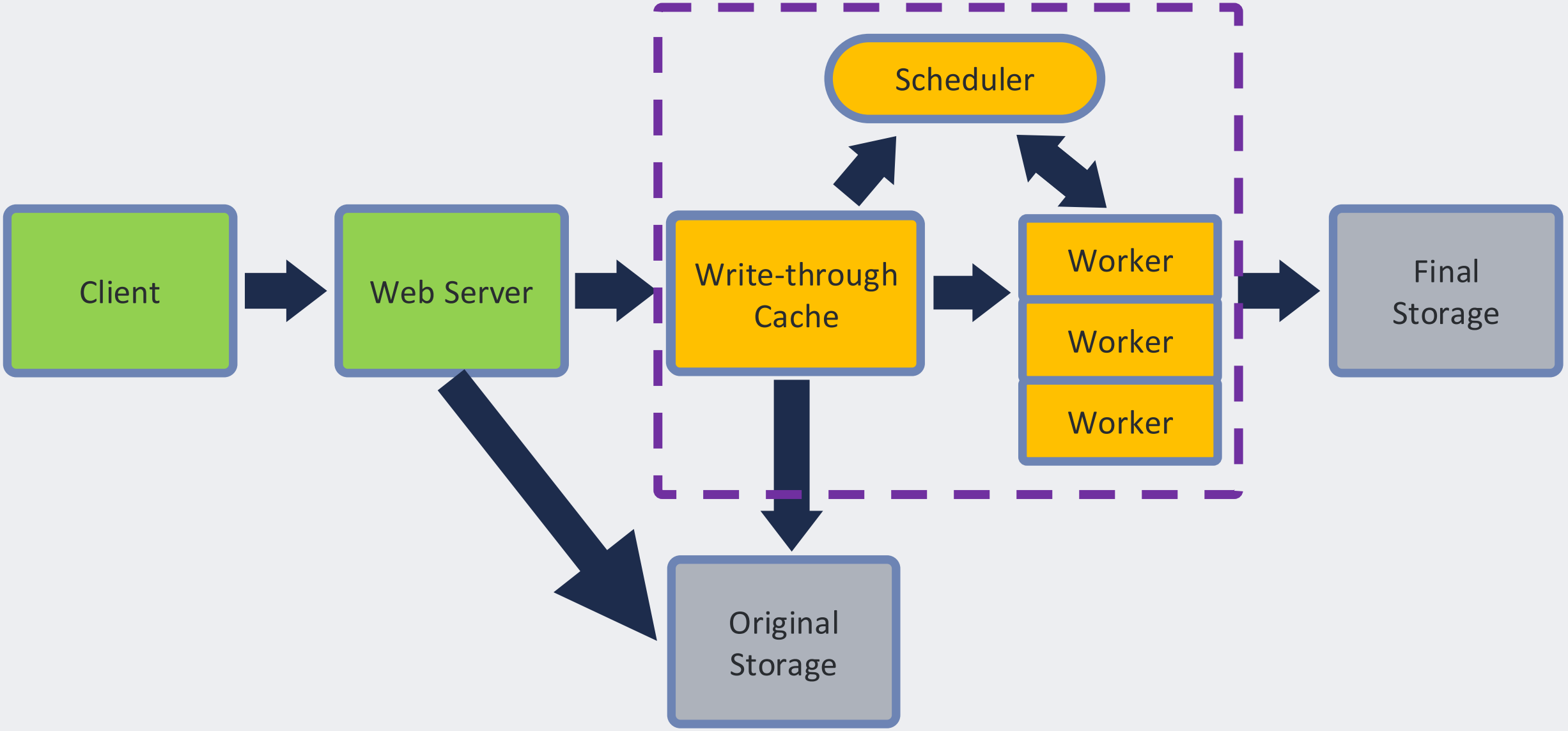
- Overlap fault tolerance and processing
- Overlap upload and processing
- Parallel processing

Architectural changes for parallelism

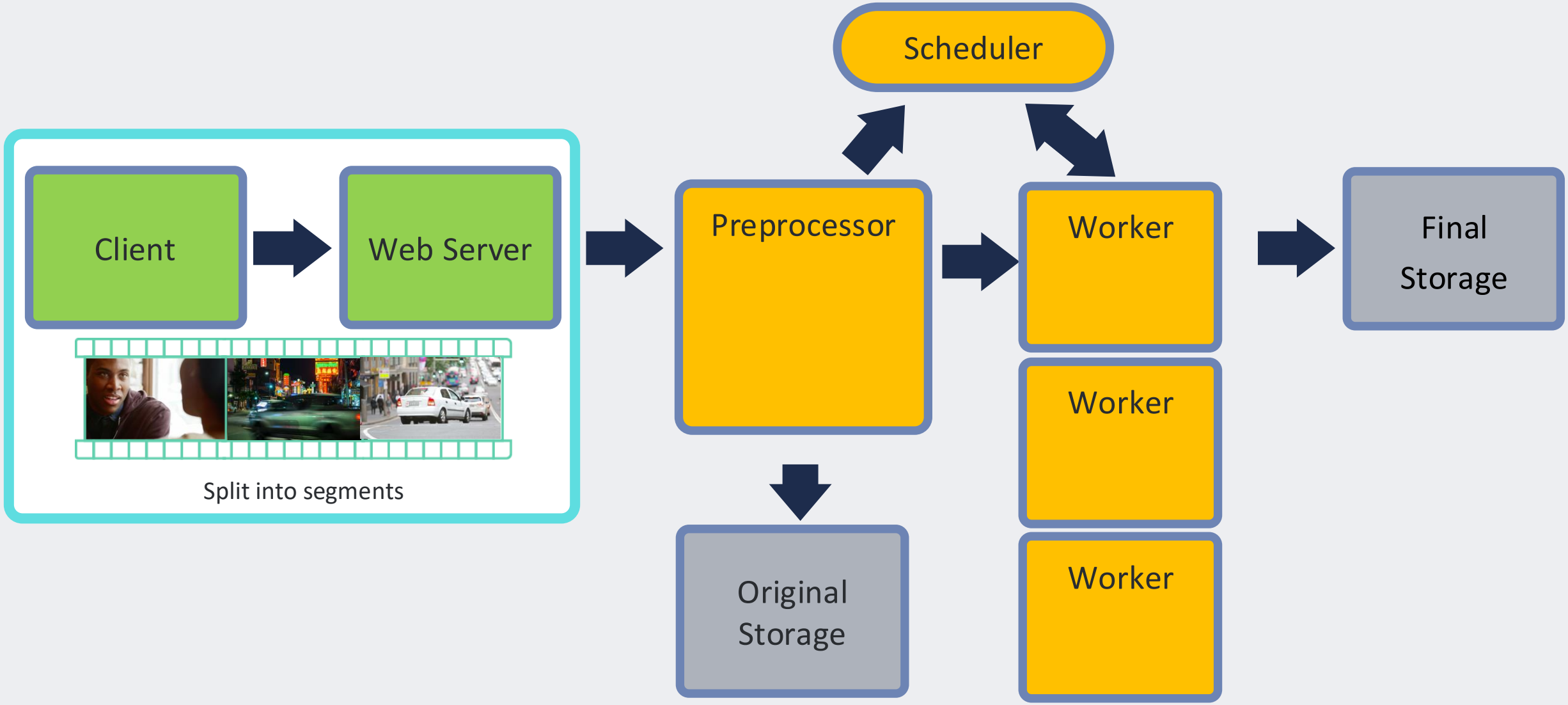
Architectural changes for parallelism



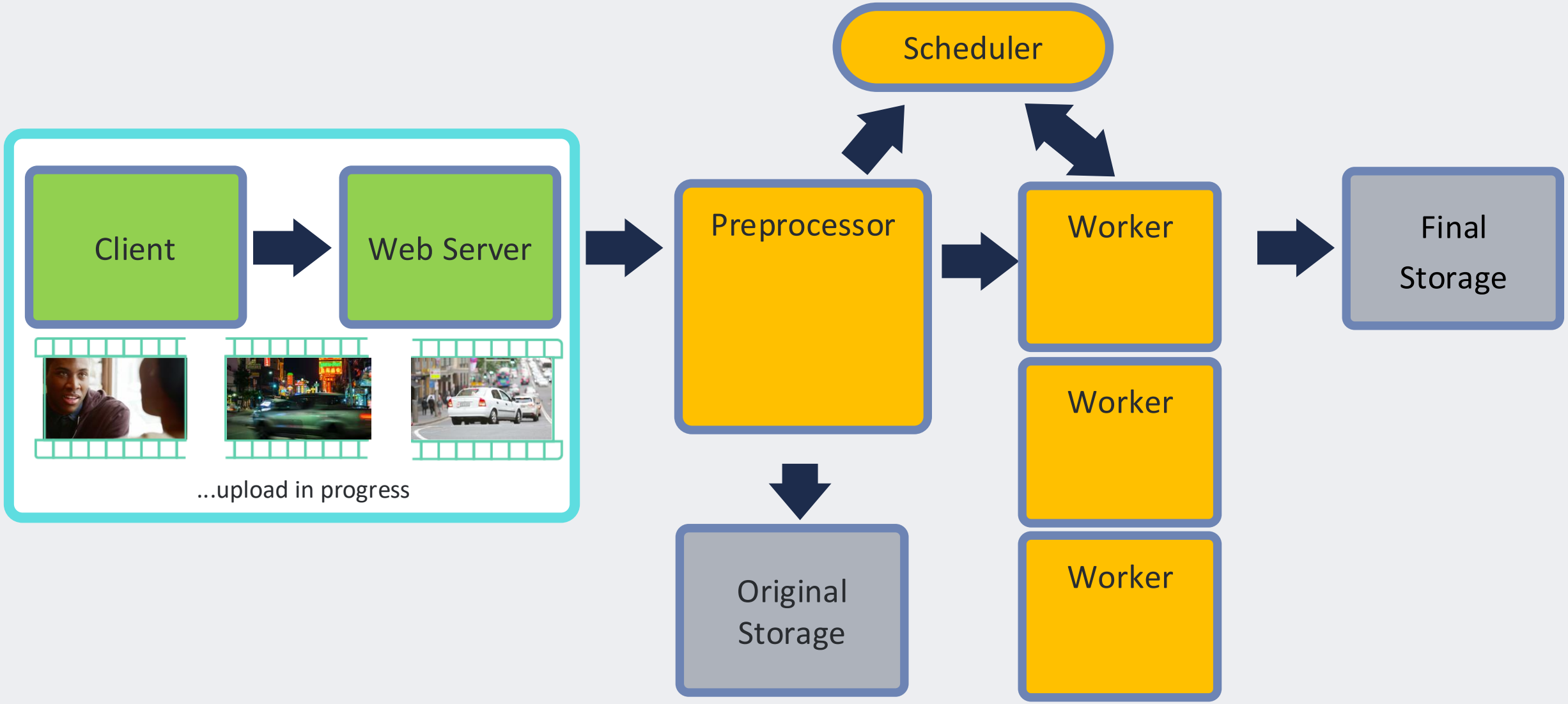
Overlap fault tolerance and processing



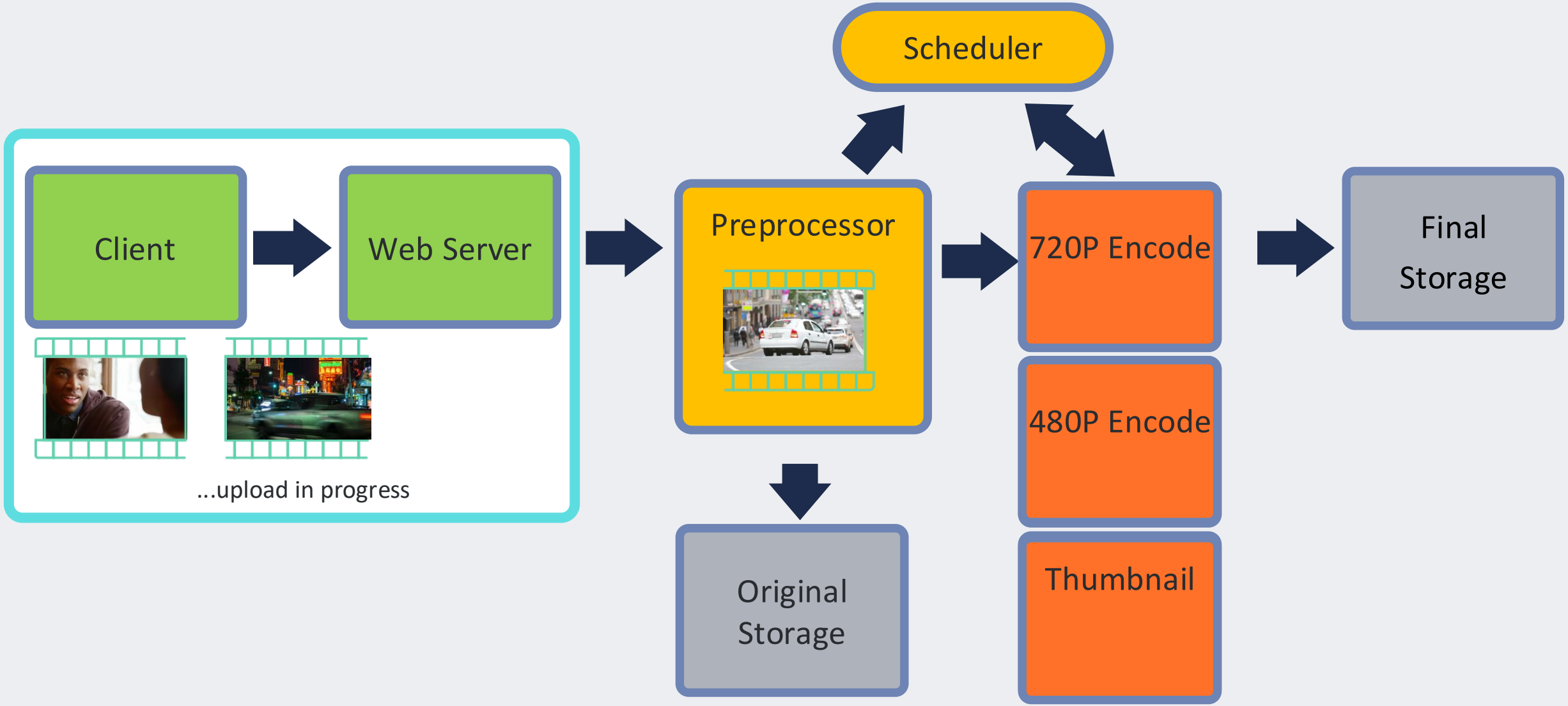
Overlap upload and processing



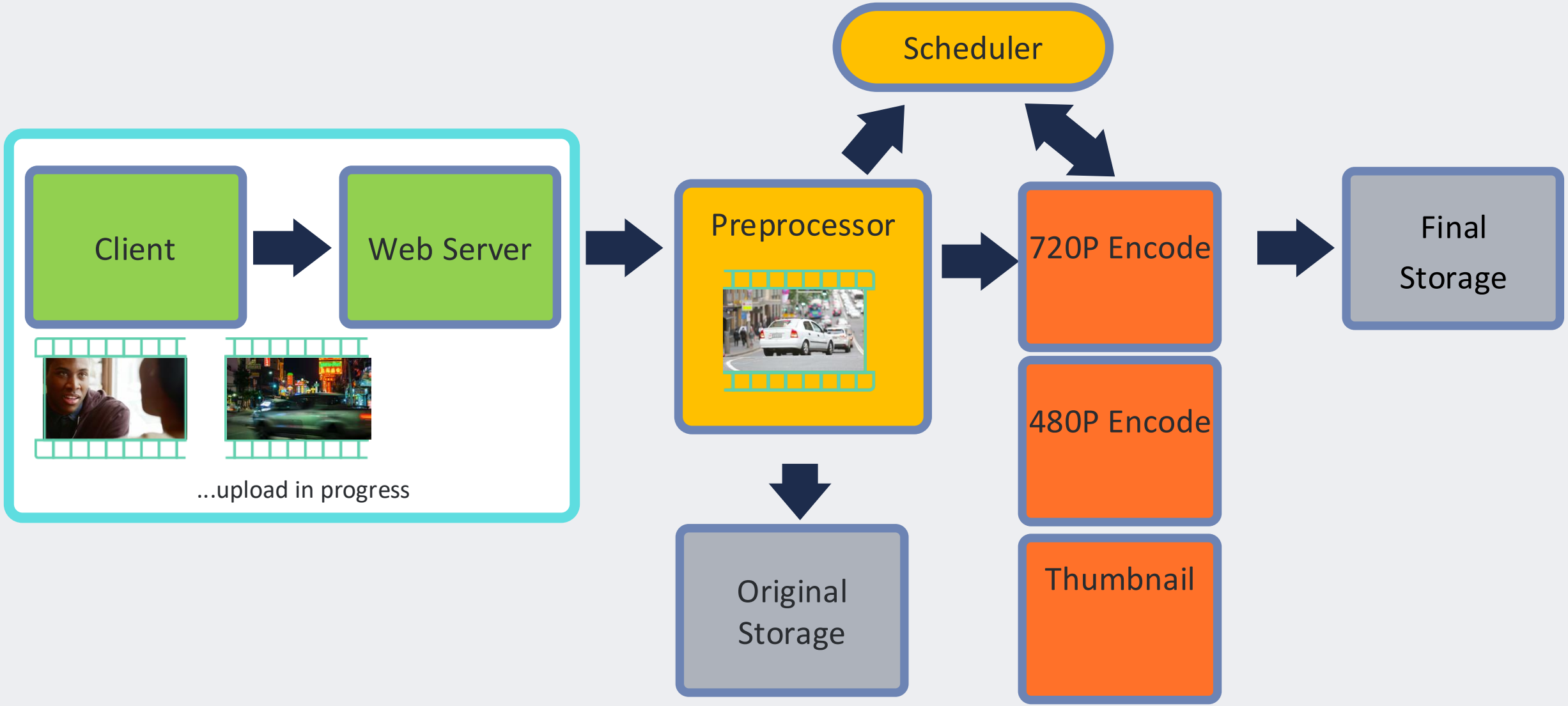
Overlap upload and processing



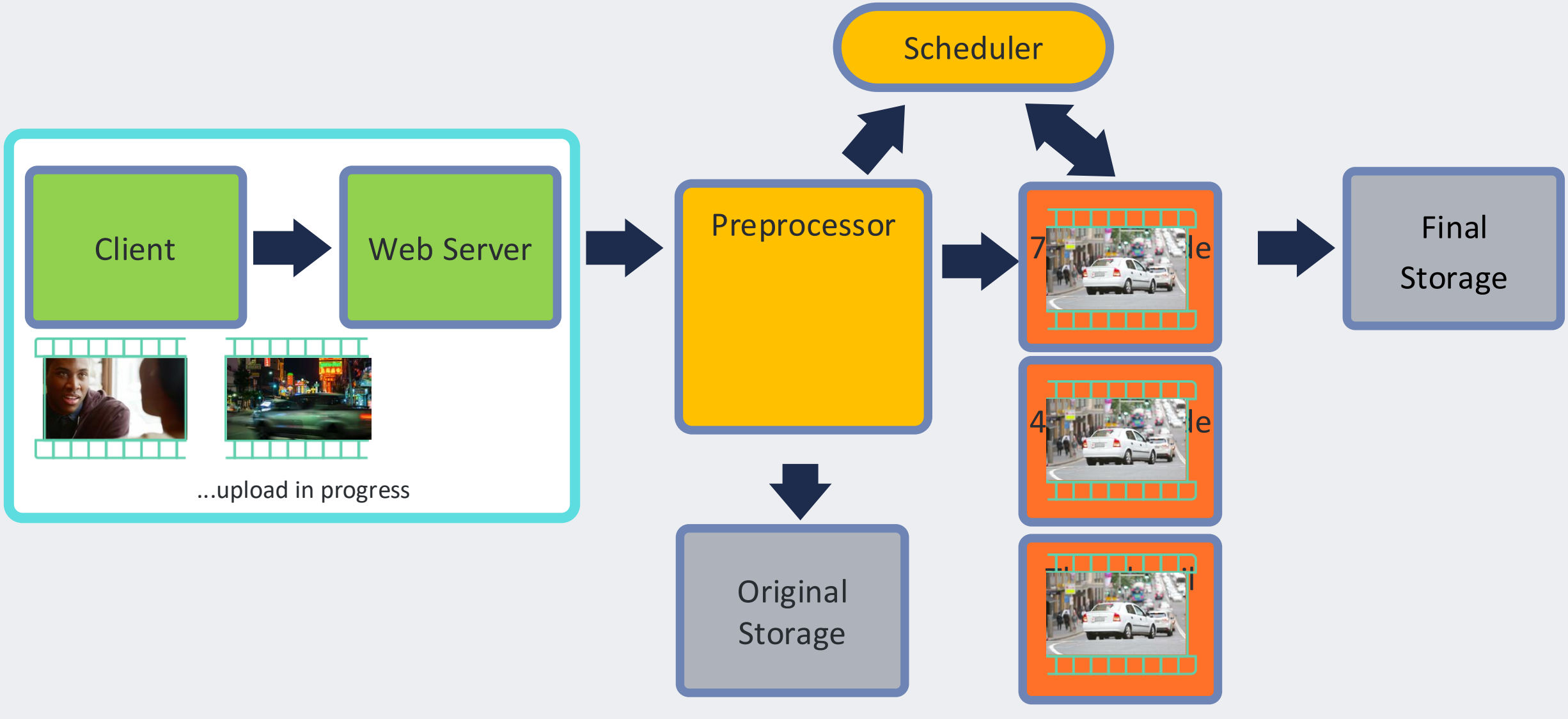
Parallel processing w/ many workers



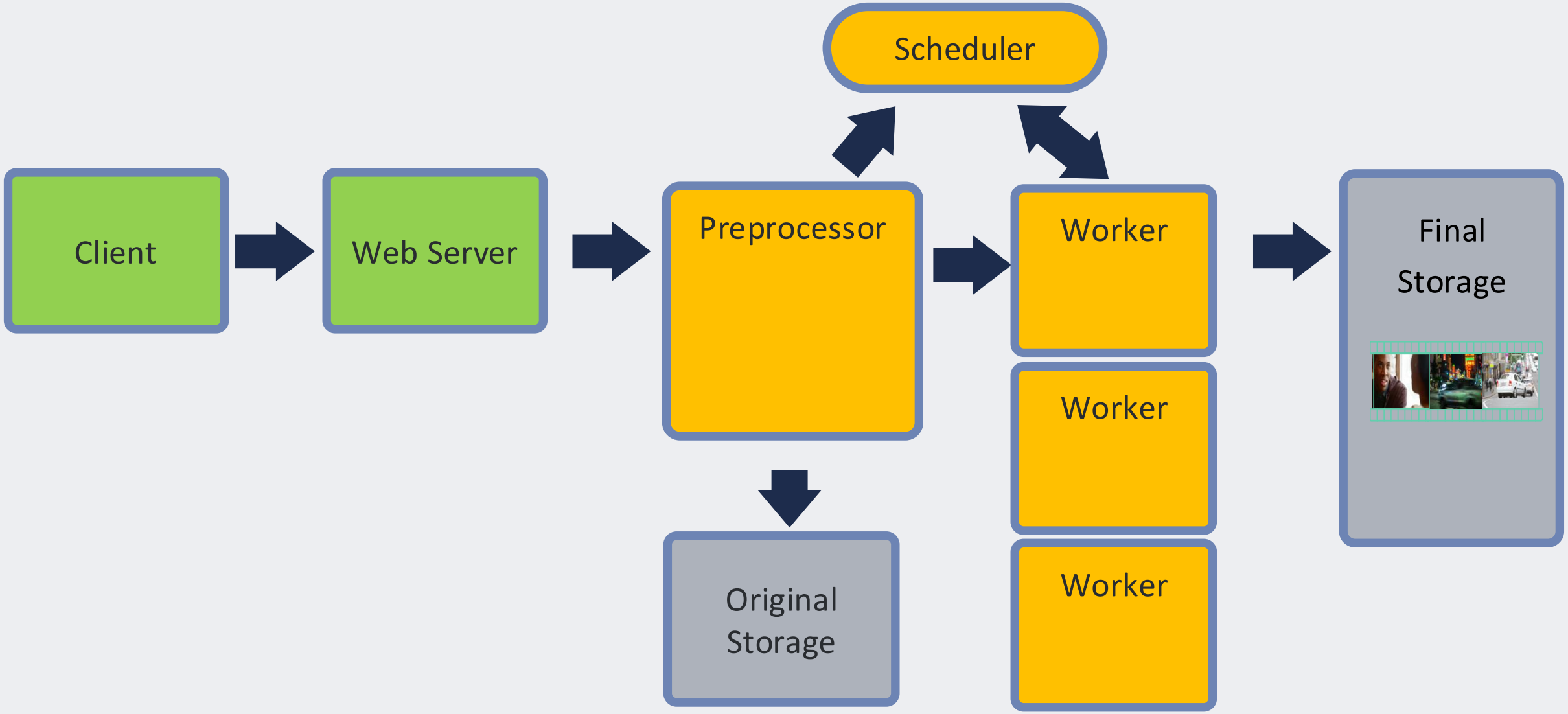
Parallel processing w/ many workers



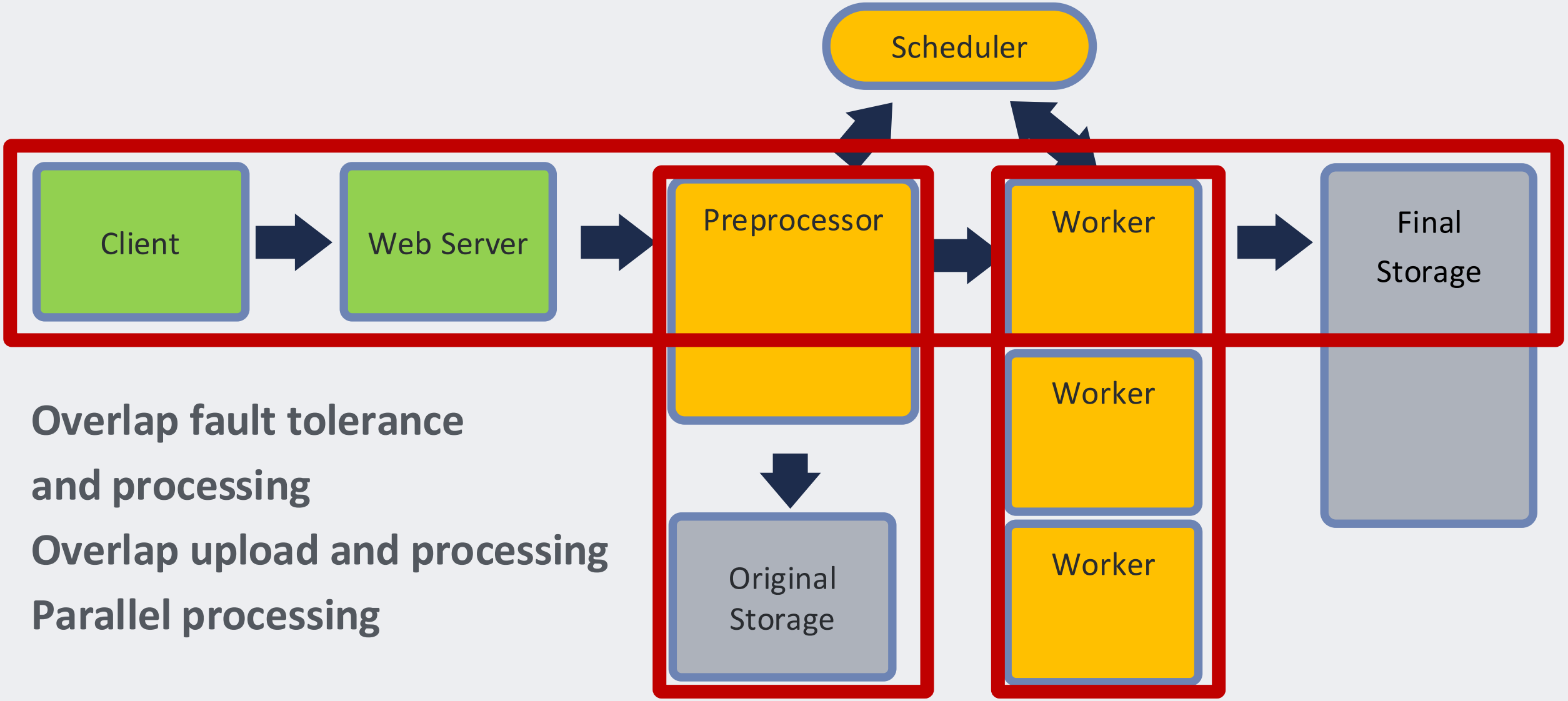
Parallel processing w/ many workers



Parallel processing w/ many workers



Three sources of parallelism

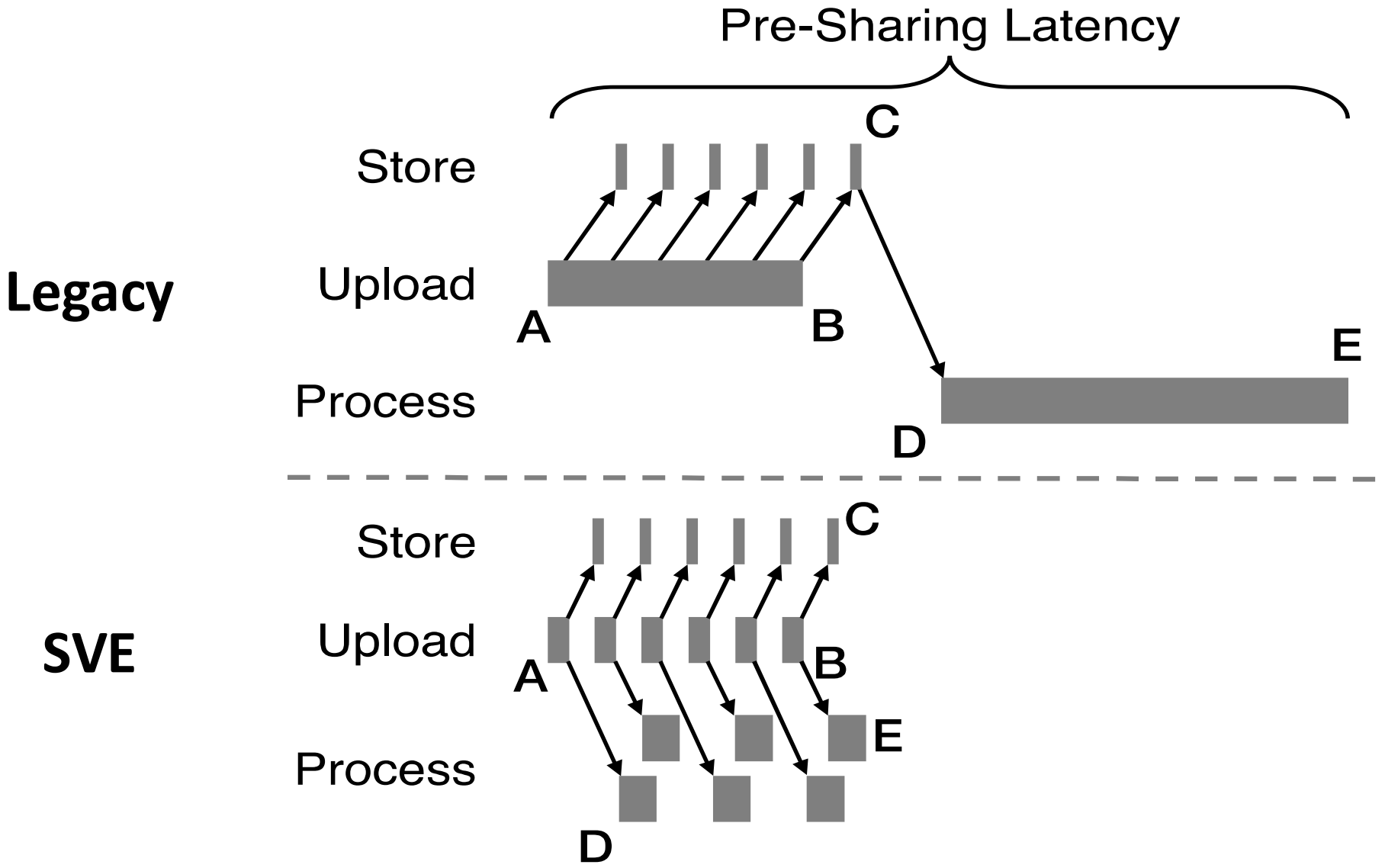


Overlap fault tolerance and processing

Overlap upload and processing

Parallel processing

Let’s Make This Faster!



7) Video now shareable with others

- Publish information about video segments to database
  - Assignment 4 – Object Relational Mapper
  - Assignment 5 – Connection pool

- Push information about video to other indexing systems
  - e.g., newsfeed on Facebook
  - e.g., subscribers on YouTube

8) User's app fetches file with metadata about video segments

- Host name -> IP address
- Global IP routing
- TCP connection
- Sockets interface
- Request routing to handler on web server

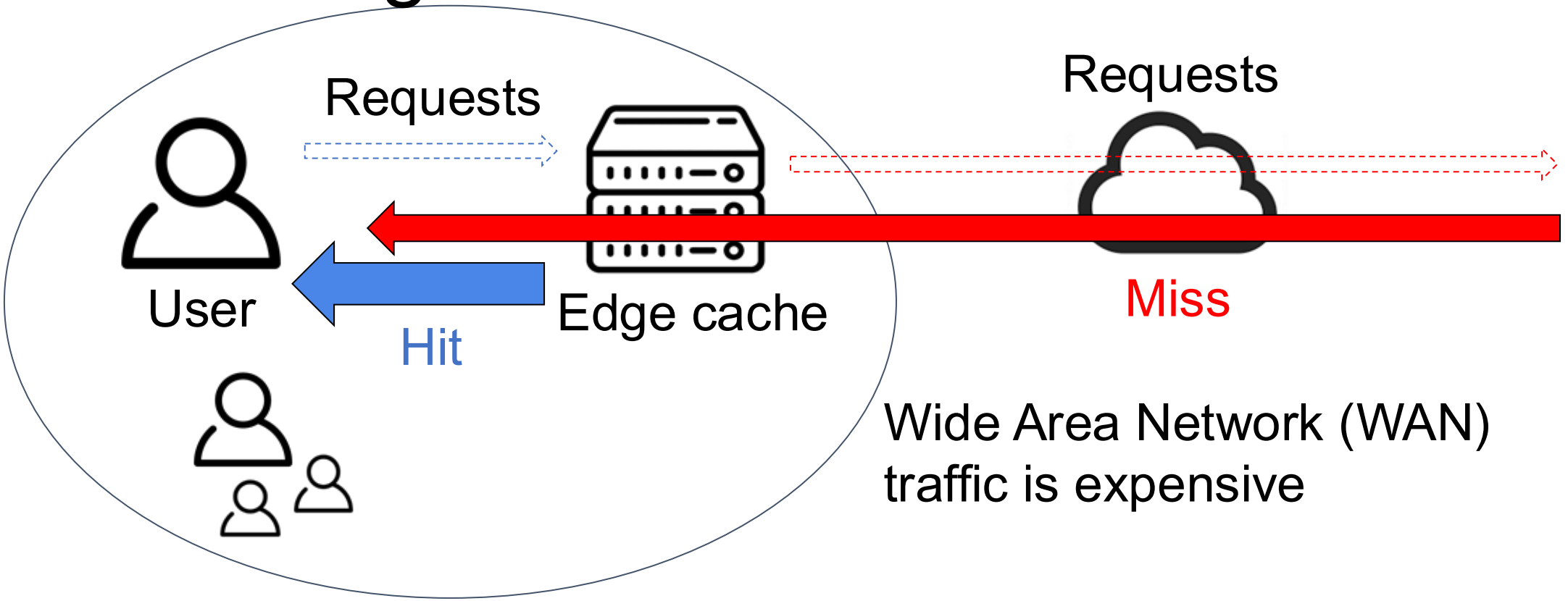
- Is user authorized to view video?

- Web server sends request to in-memory cache for video segment metadata
  - Assignment 3 – Caching!

9) User’s app runs ABR algorithm to download video segments via CDN

- Adaptive BitRate (ABR) algorithm request video segments
- Video segment requests via Content Distribution Network
- CDNs cache popular video segments

CDN Caching Goal: Minimize WAN Traffic



Wide Area Network (WAN) traffic is expensive

Key metric hit ratio

Caching Remains Challenging

Heuristic-based algorithms (1965–): **LRU**, **LFU**, GDSF, ARC, ...

- Work well for some workloads, but work poorly for other

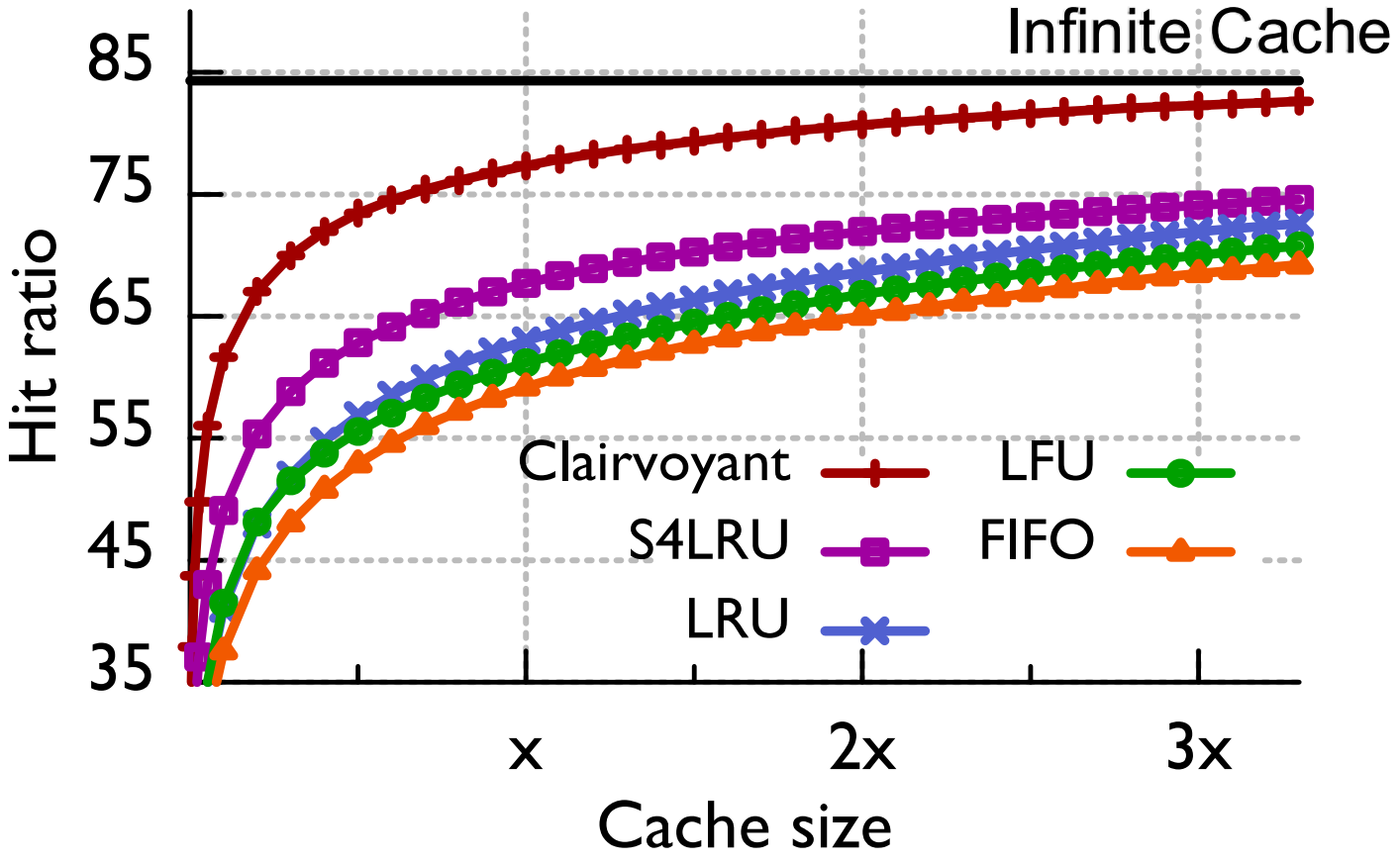
ML-based adaptation of heuristics (2017–): UCB, LeCAR, ...

- Also work well for some workloads, but poorly for others

The **Belady** algorithm (1966)

- Offline optimal: requires future knowledge
- Large gap in miss ratio between state-of-the-art and Belady:
- 20–40% on production traces

Edge Cache with Different Algos



- Clairvoyant (Bélády) shows we can do much better!

Research From Princeton!

Learning Relaxed Belady for
Content Distribution Network Caching.

Zhenyu Song, Daniel S. Berger,
Kai Li, and Wyatt Lloyd.

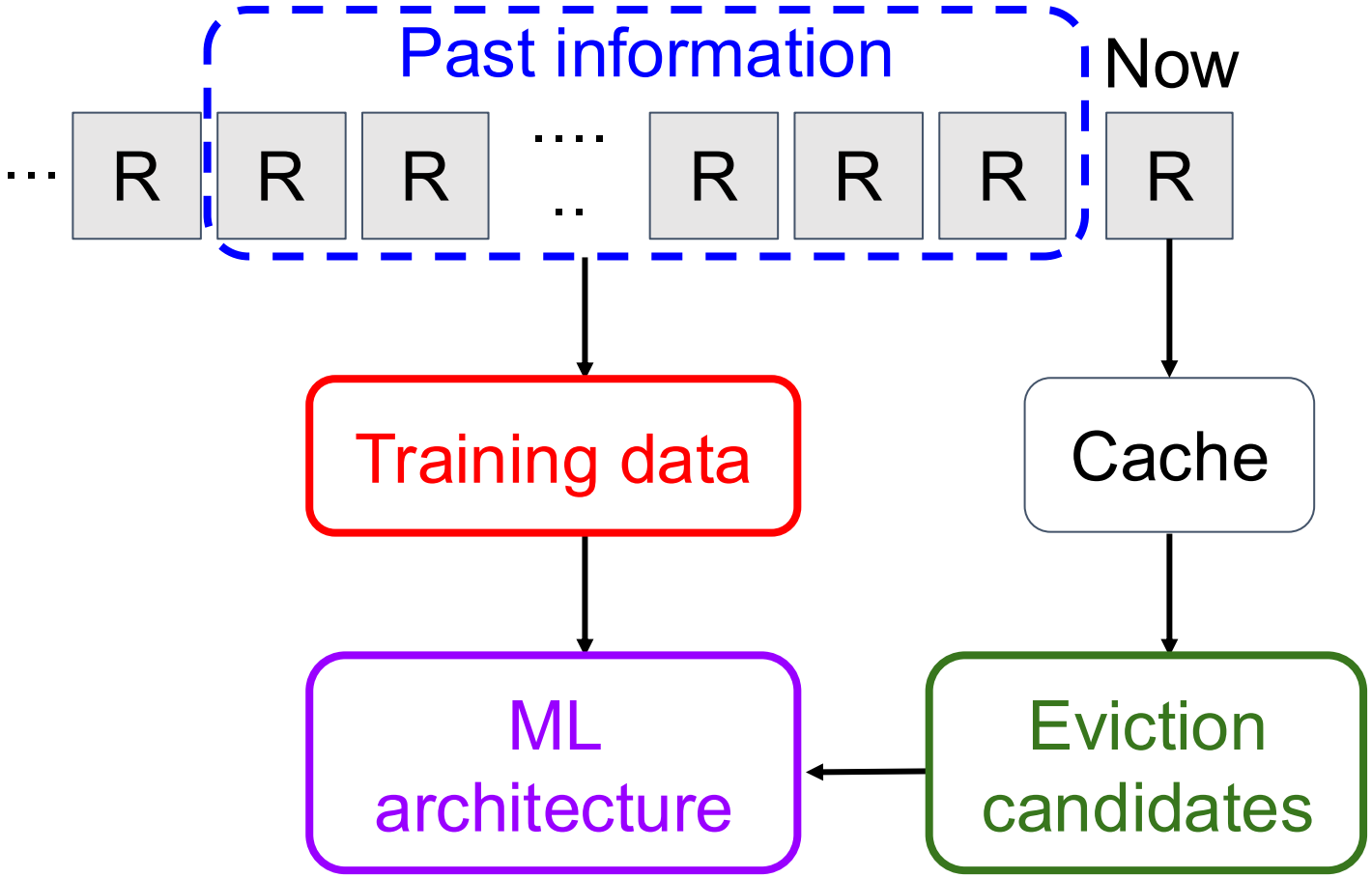
In 17th USENIX Symposium on
Networked Systems Design and
Implementation (NSDI 20), February
2020.

Introducing Learning Relaxed Belady (LRB)

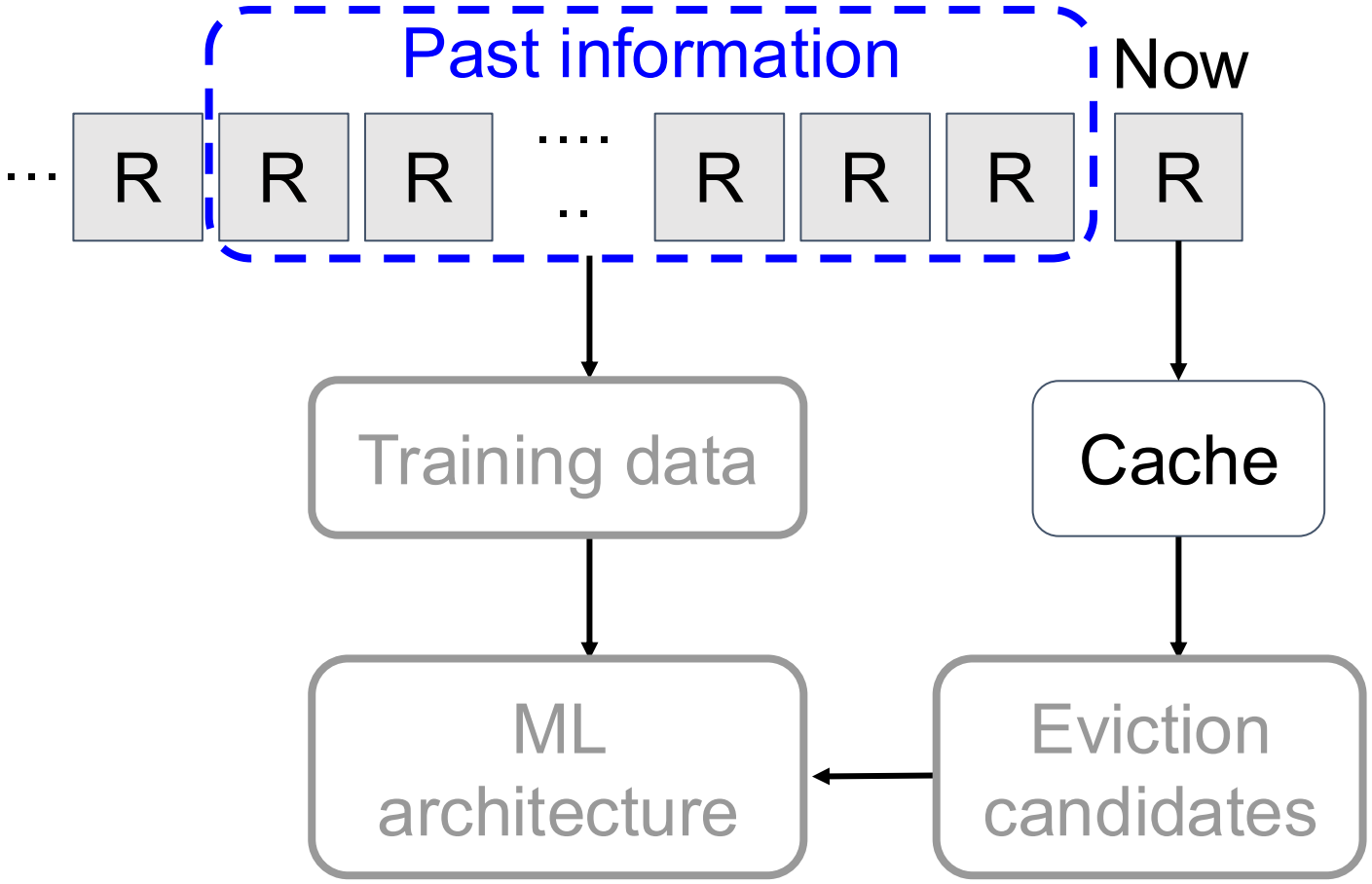
New approach: mimic Belady using machine learning

- Machine-Learning-for-Systems (ML-for-Systems)
  - Enabling technologies
  - When does it make sense?

General Overview of our Approach



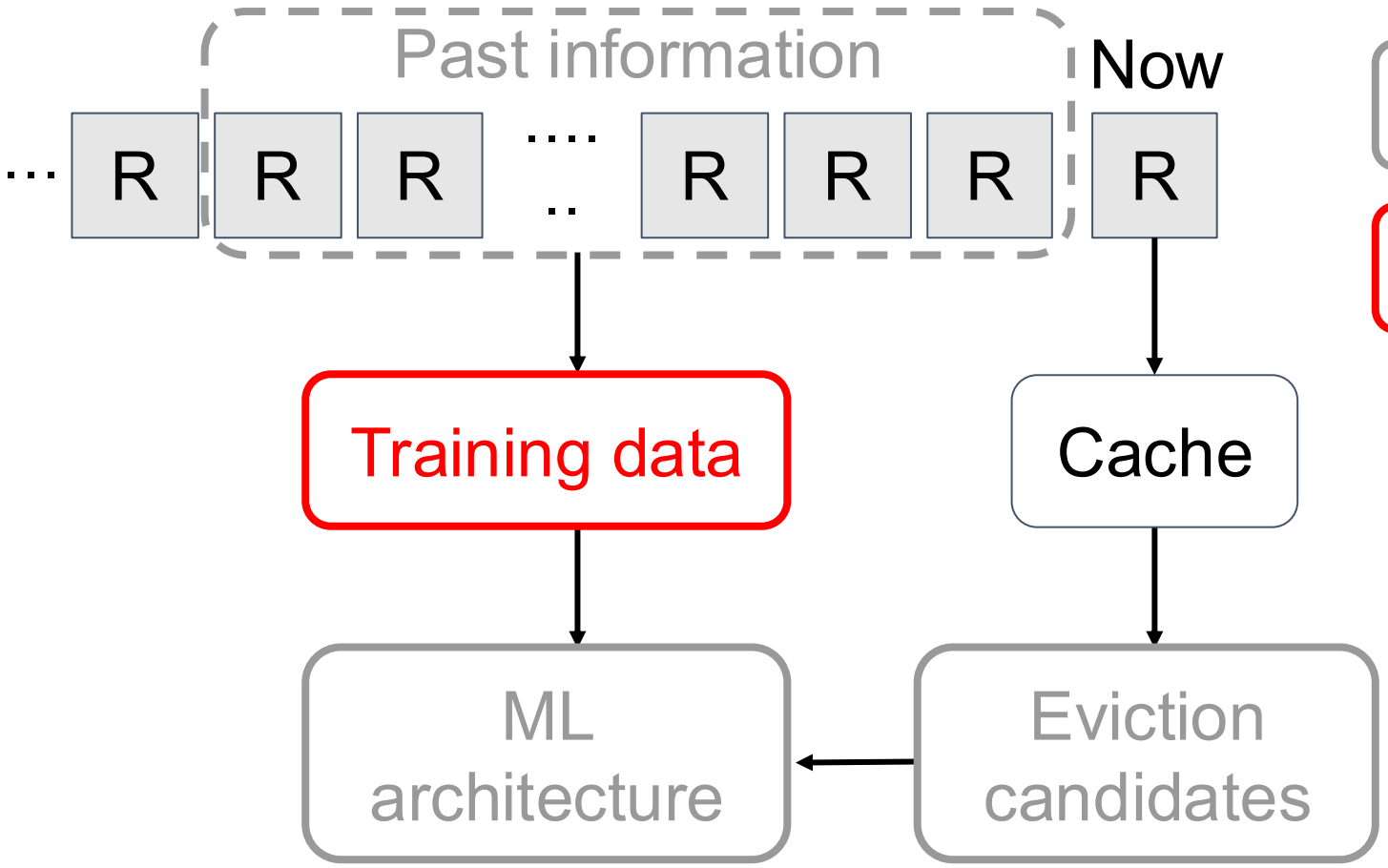
Challenge 1: Past Information



What past information to use?

More data improves training but increases memory overhead

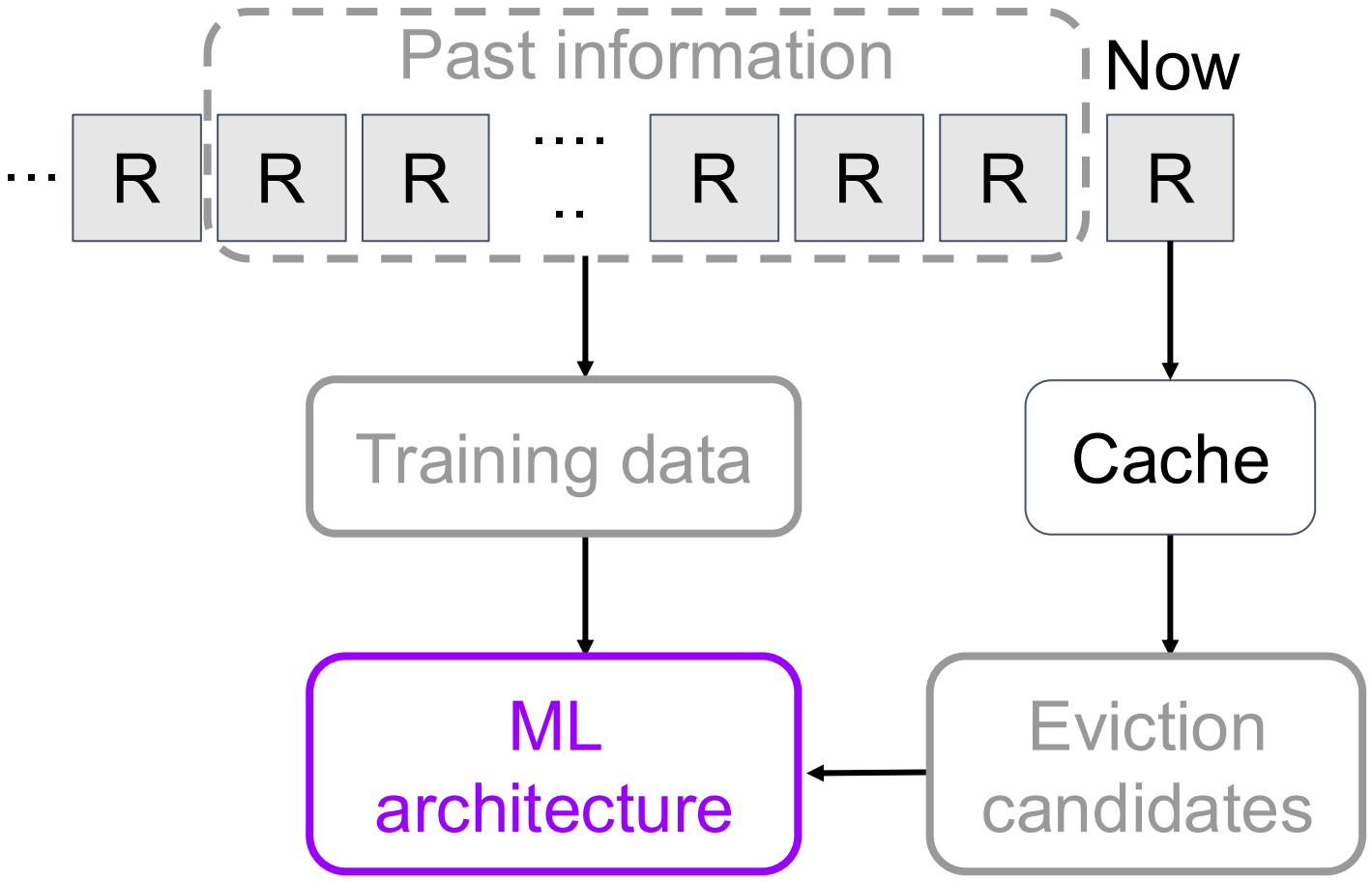
Challenge 2: Generate Online Training Data



What past information to use?

Generate online training data?

Challenge 3: ML Architecture



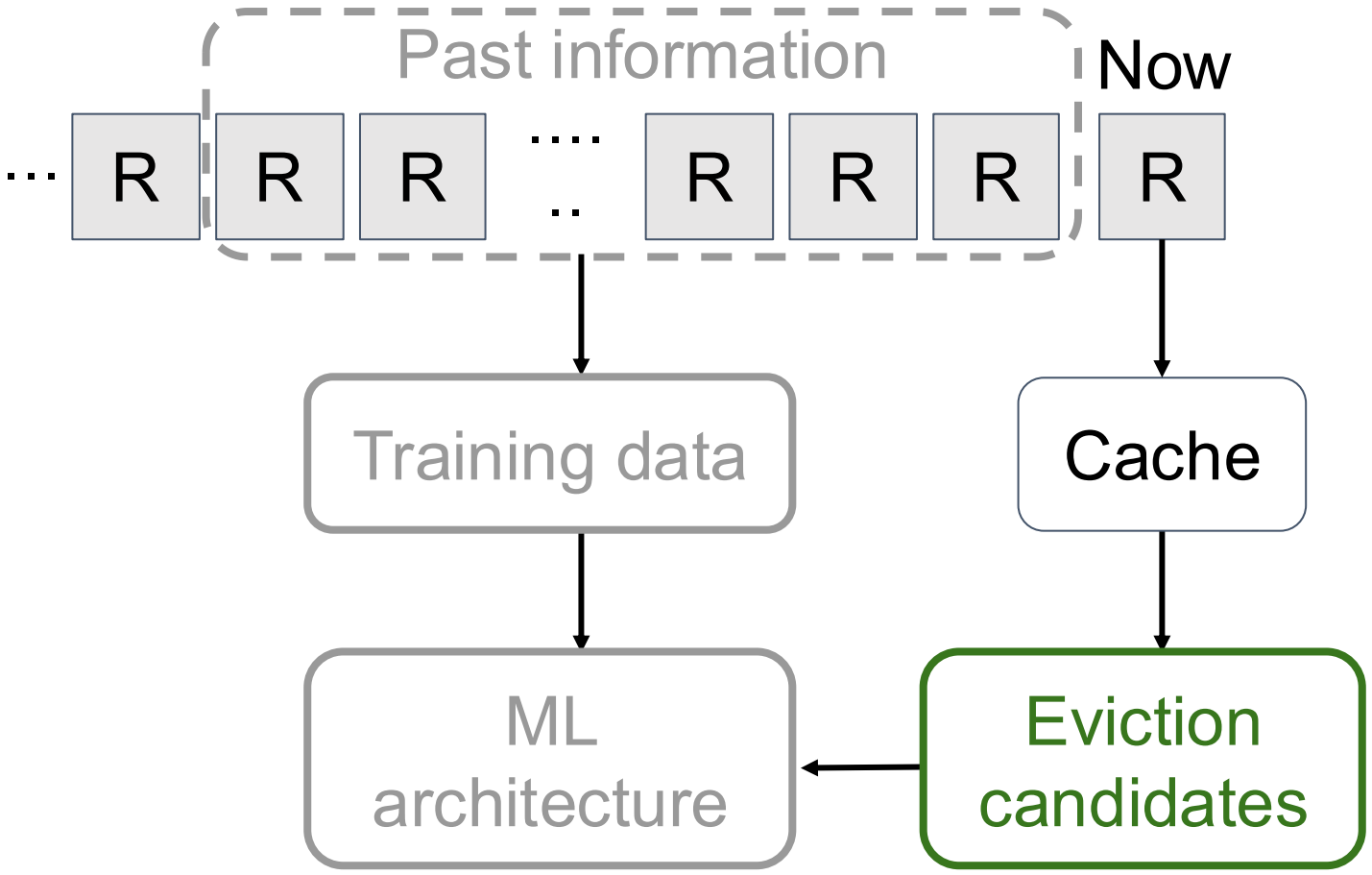
What past information to use?

Generate online training data?

What ML architecture to select?

Large design space: features, model, prediction target, loss function

Challenge 4: Eviction Candidates



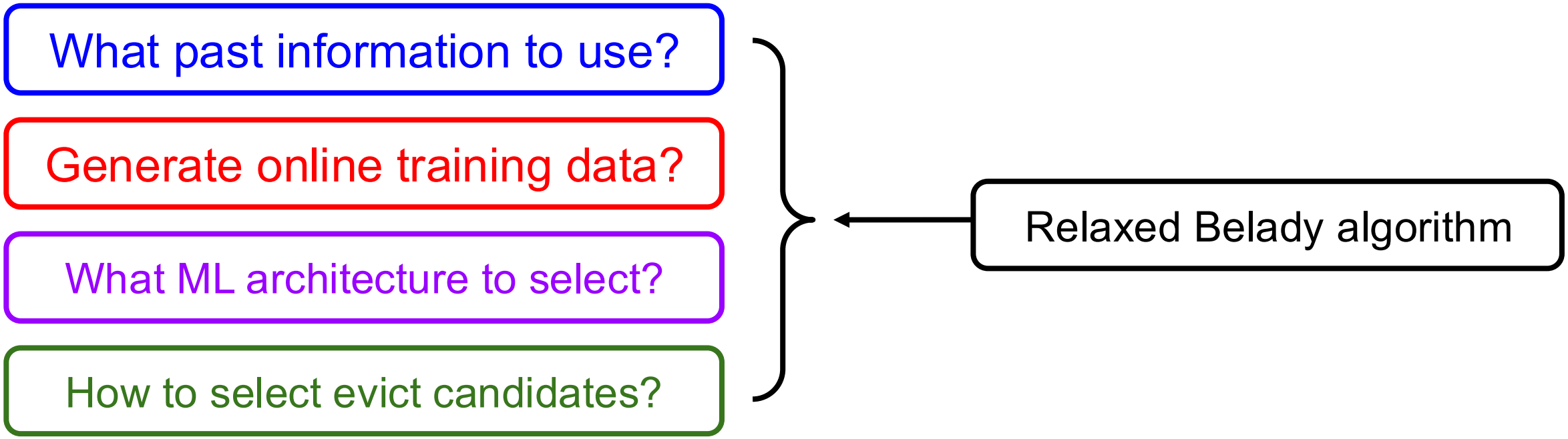
What past information to use?

Generate online training data?

What ML architecture to select?

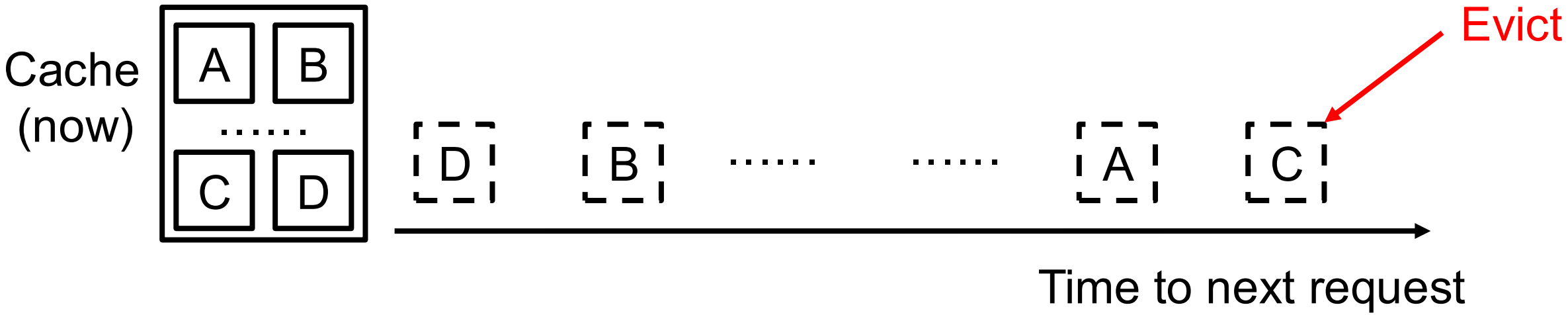
How to select evict candidates?

Solution: Relaxed Belady Algorithm



Challenge: Hard to Mimic Belady Algorithm

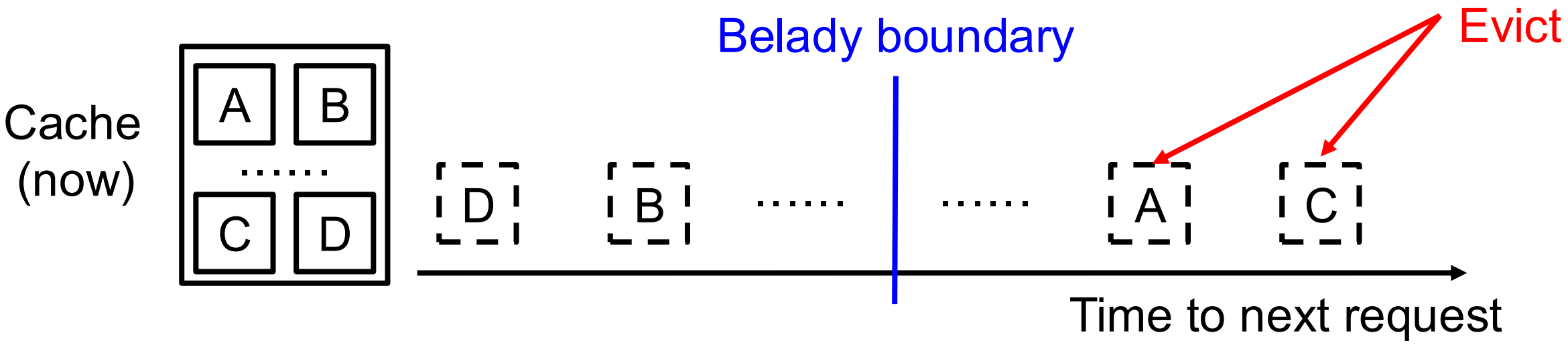
Belady: evict object with next access farthest in the future



Mimicking exact Belady is impractical

- Need predictions for all objects → prohibitive computational cost
- Need exact prediction of next access → further prediction are harder

Introducing the Relaxed Belady Algorithm

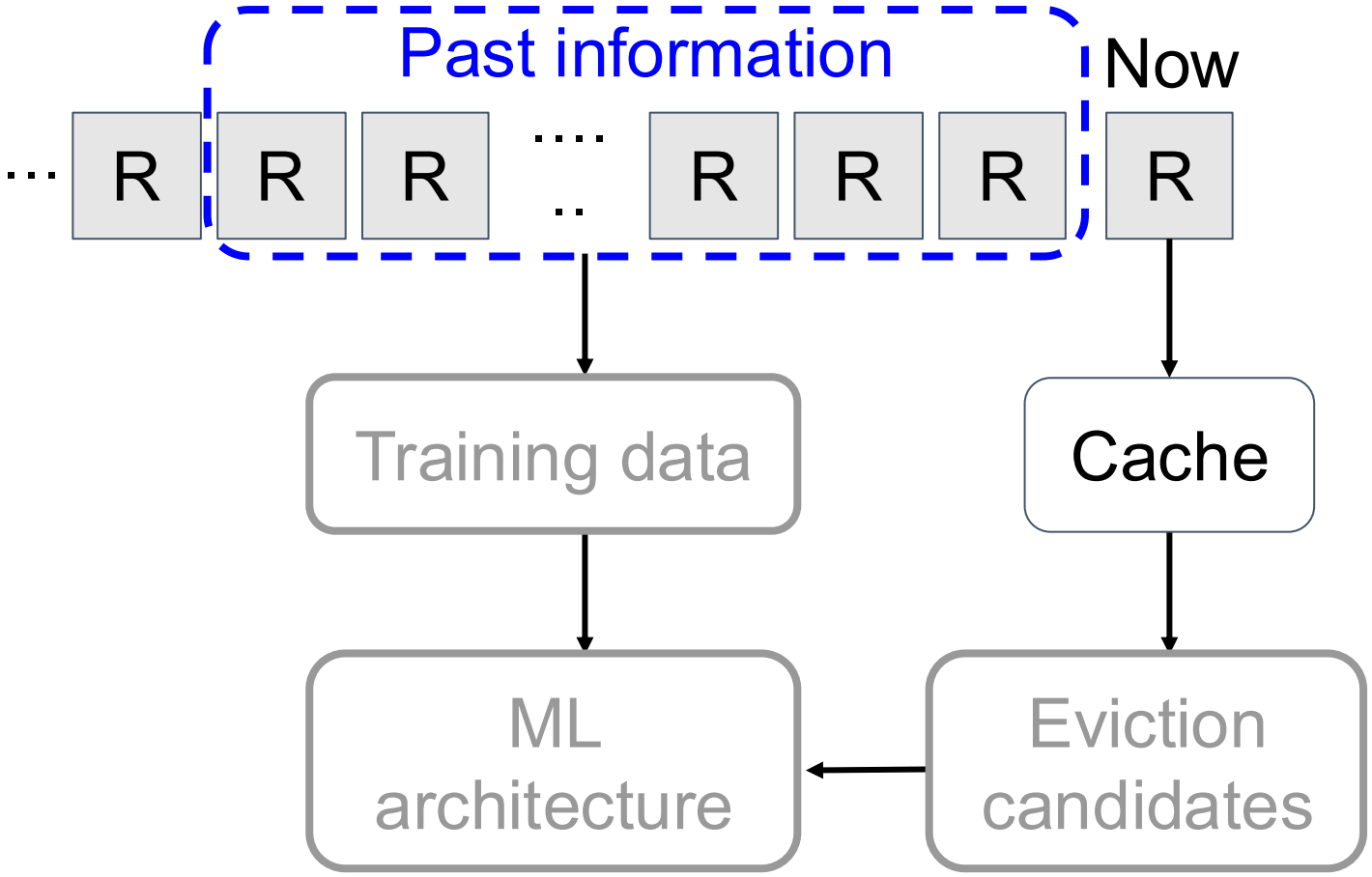


Observation: many objects are good candidates for eviction

Relaxed Belady evicts a random object beyond boundary

- Do not need predictions for all objects → reasonable computation
- No need to differentiate beyond boundary → simplifies the prediction

Challenge 1: Past Information

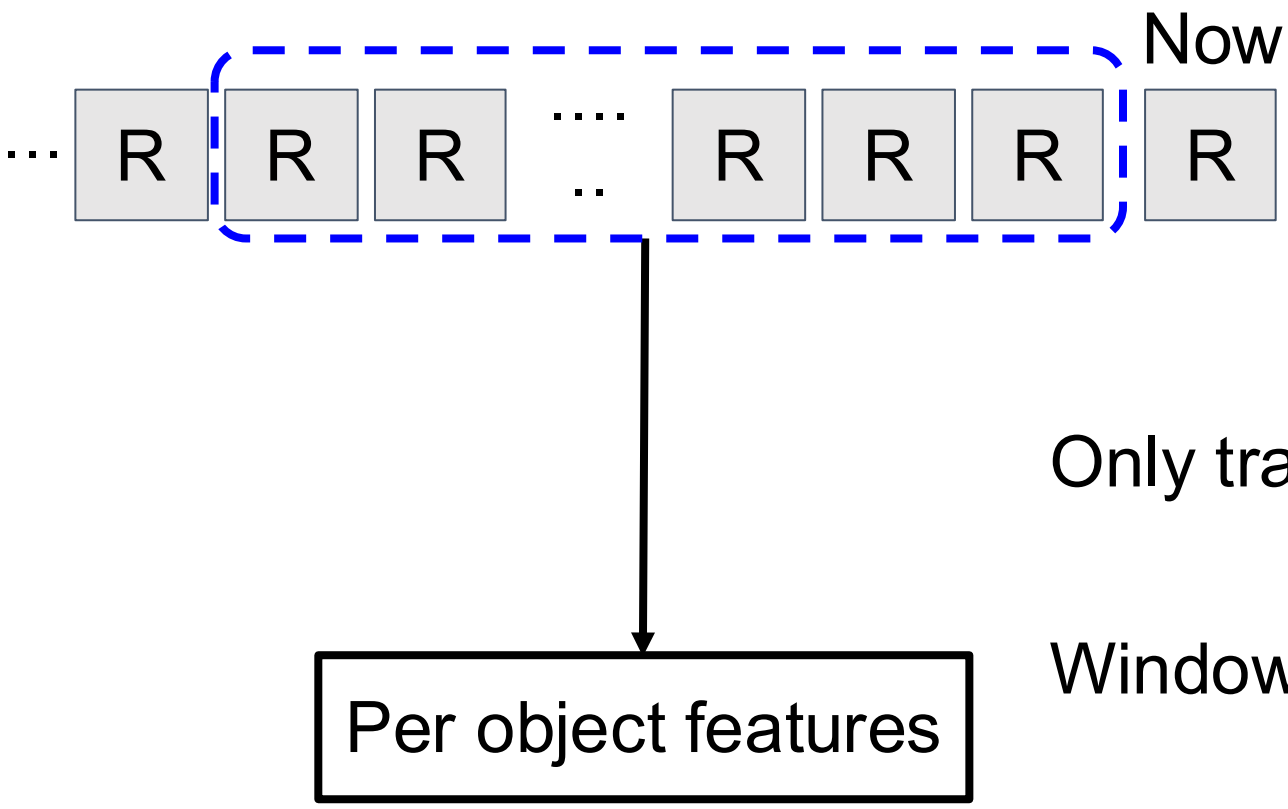


What past information to use?

More data improves training but increases memory overhead

Track Objects within a Sliding Memory Window

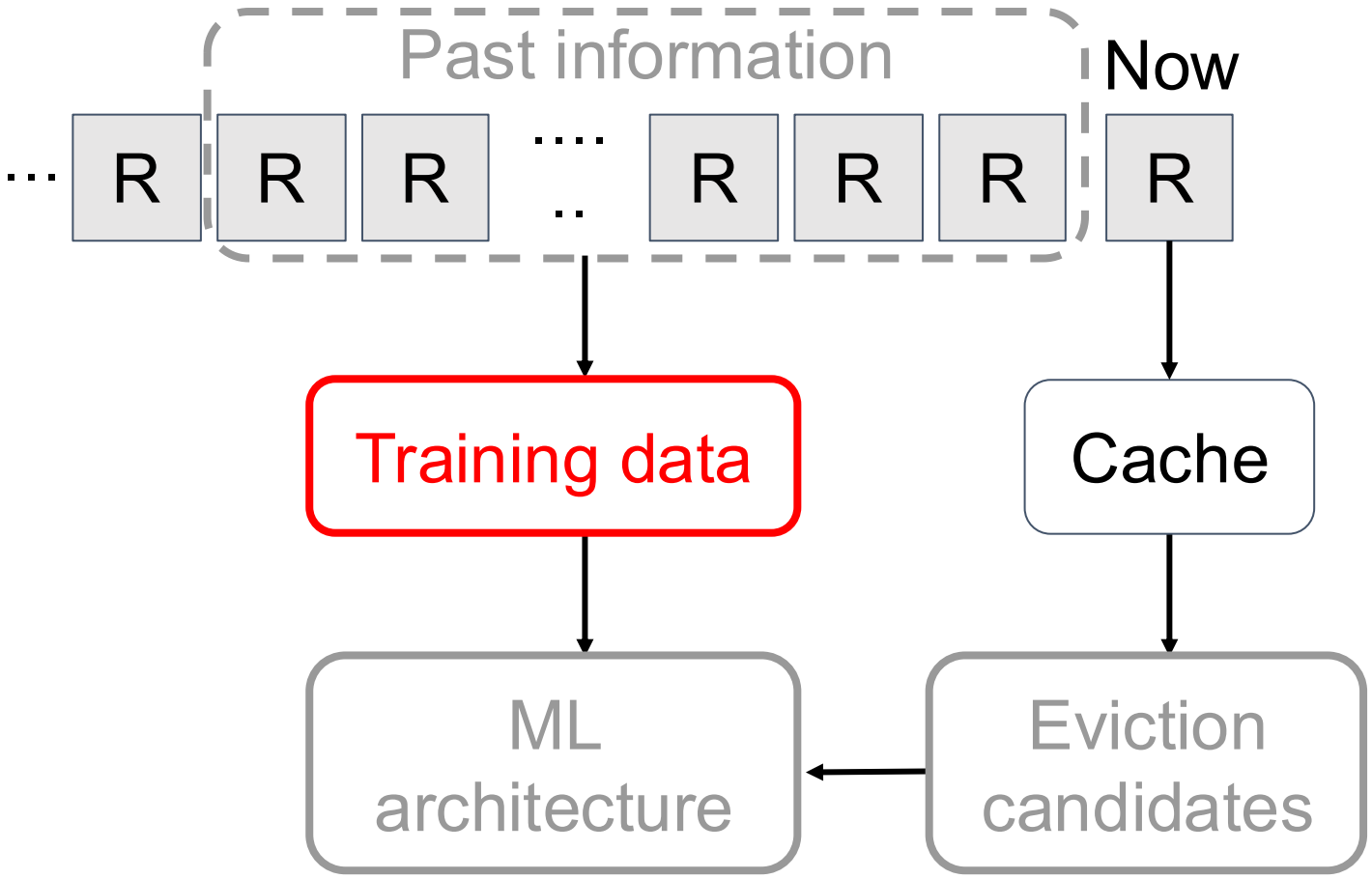
Sliding memory window mimics Belady boundary



Only track objects within memory window

Window size is LRB's main hyperparameter

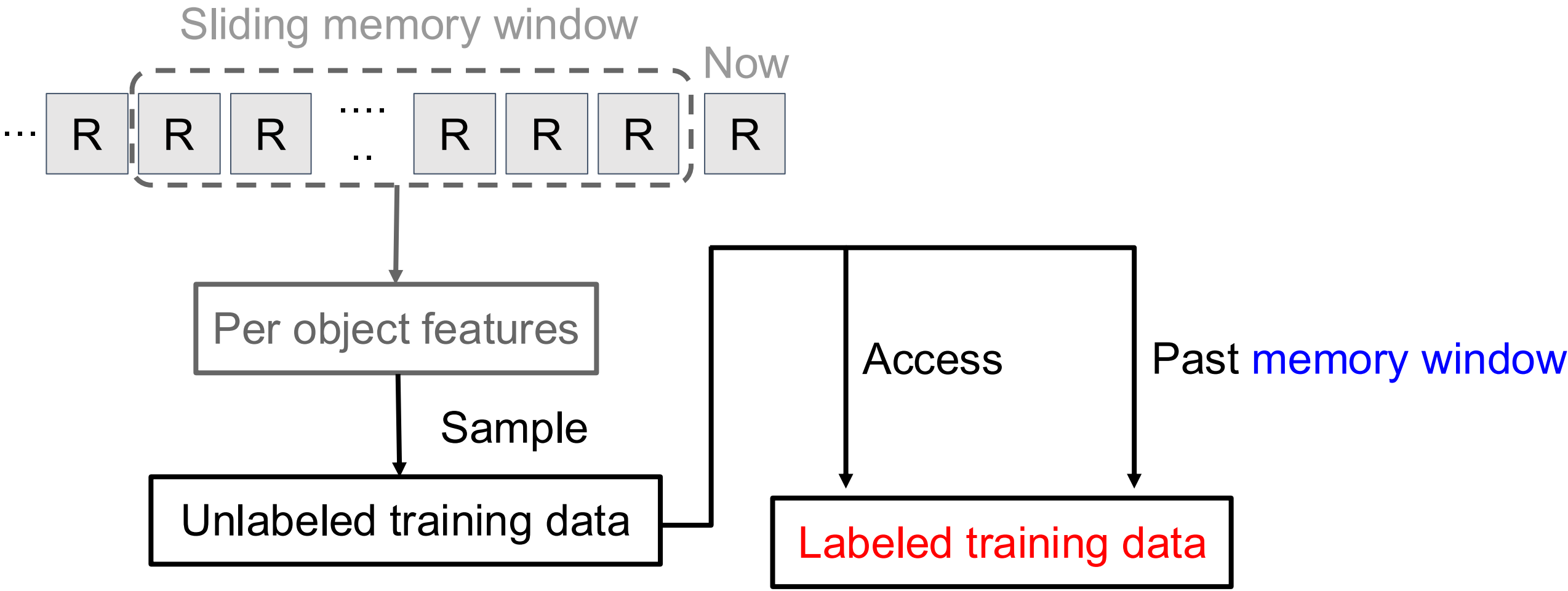
Challenge 2: Training Data



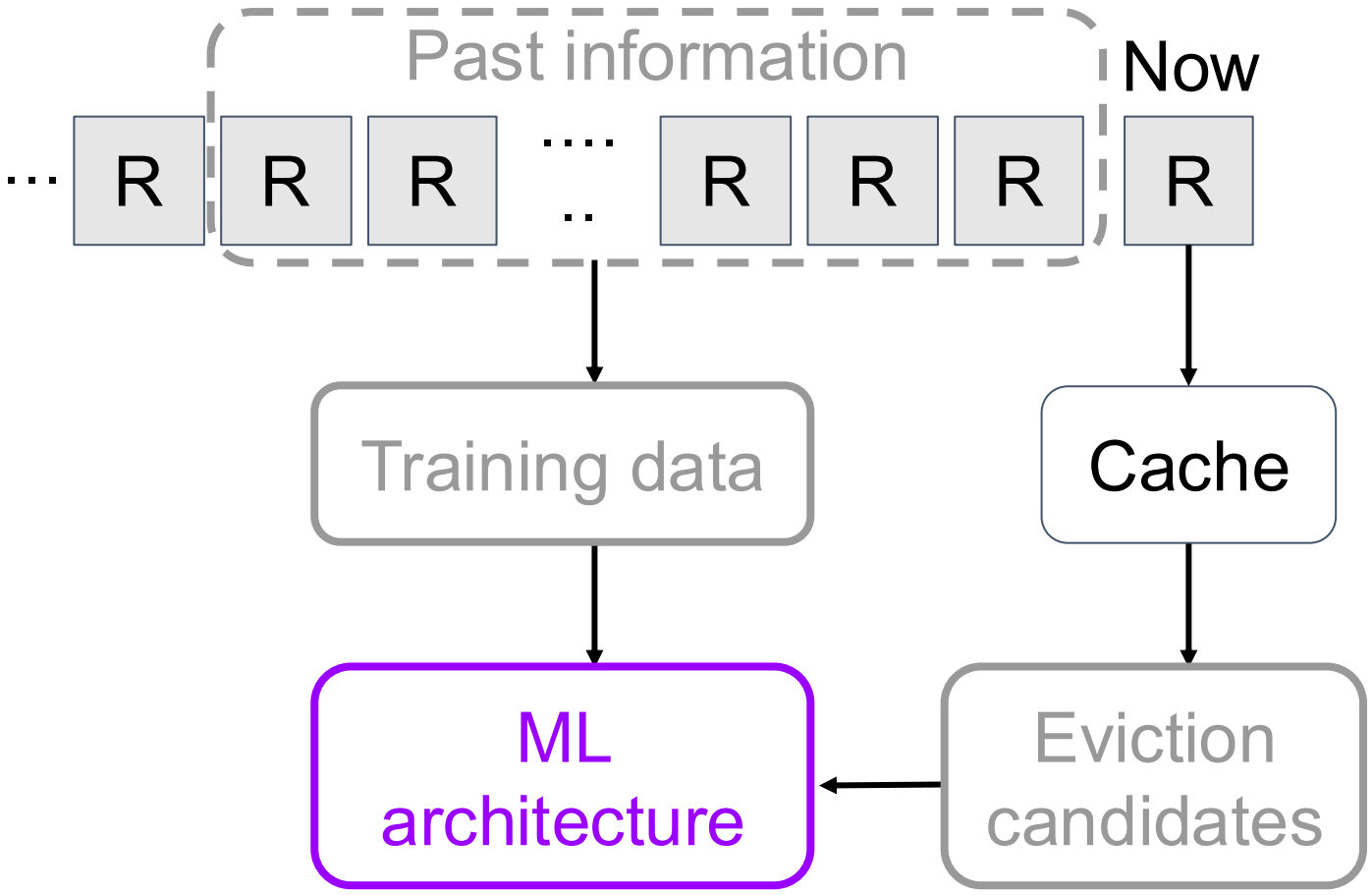
What past information to use?

Generate online training data?

Sample Training Data & Label on Access or Boundary



Challenge 3: ML Architecture



What past information to use?

Generate online training data?

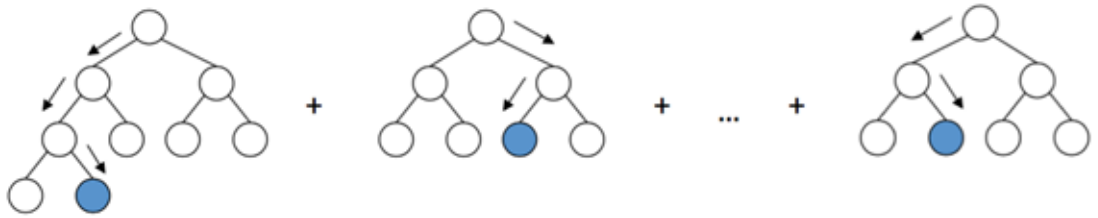
What ML architecture to select?

Large potential design space

Solution 3: Feature & Model Selection

Use good decision ratio to evaluate new designs

Features
Object size
Object type
Inter-request distances (recency)
Exponential decay counters (long-term frequencies)

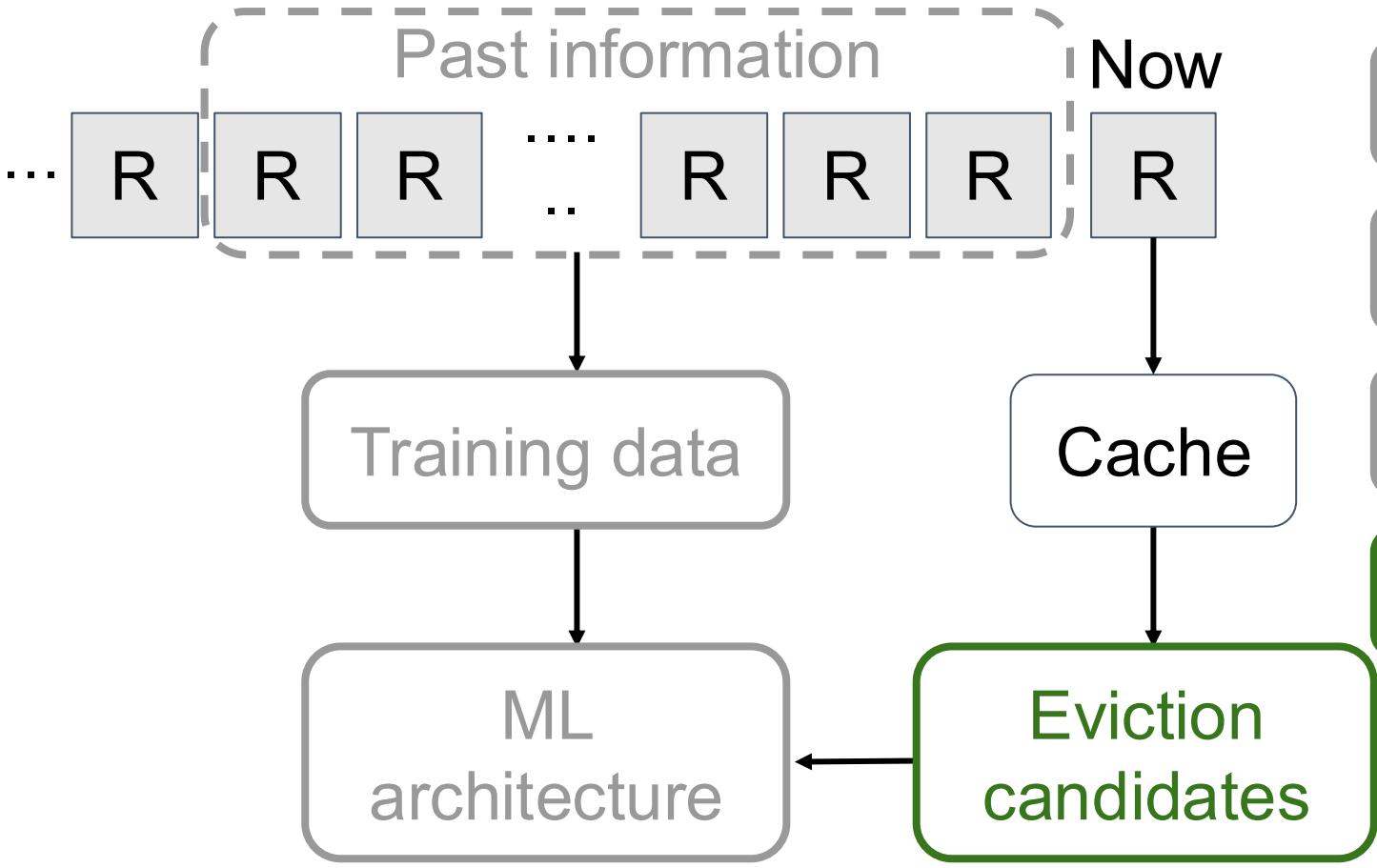


Gradient boosting decision trees

Lightweight & **high good decision ratio**

Training ~300 ms, prediction ~30 us

Challenge 4: Eviction Candidates



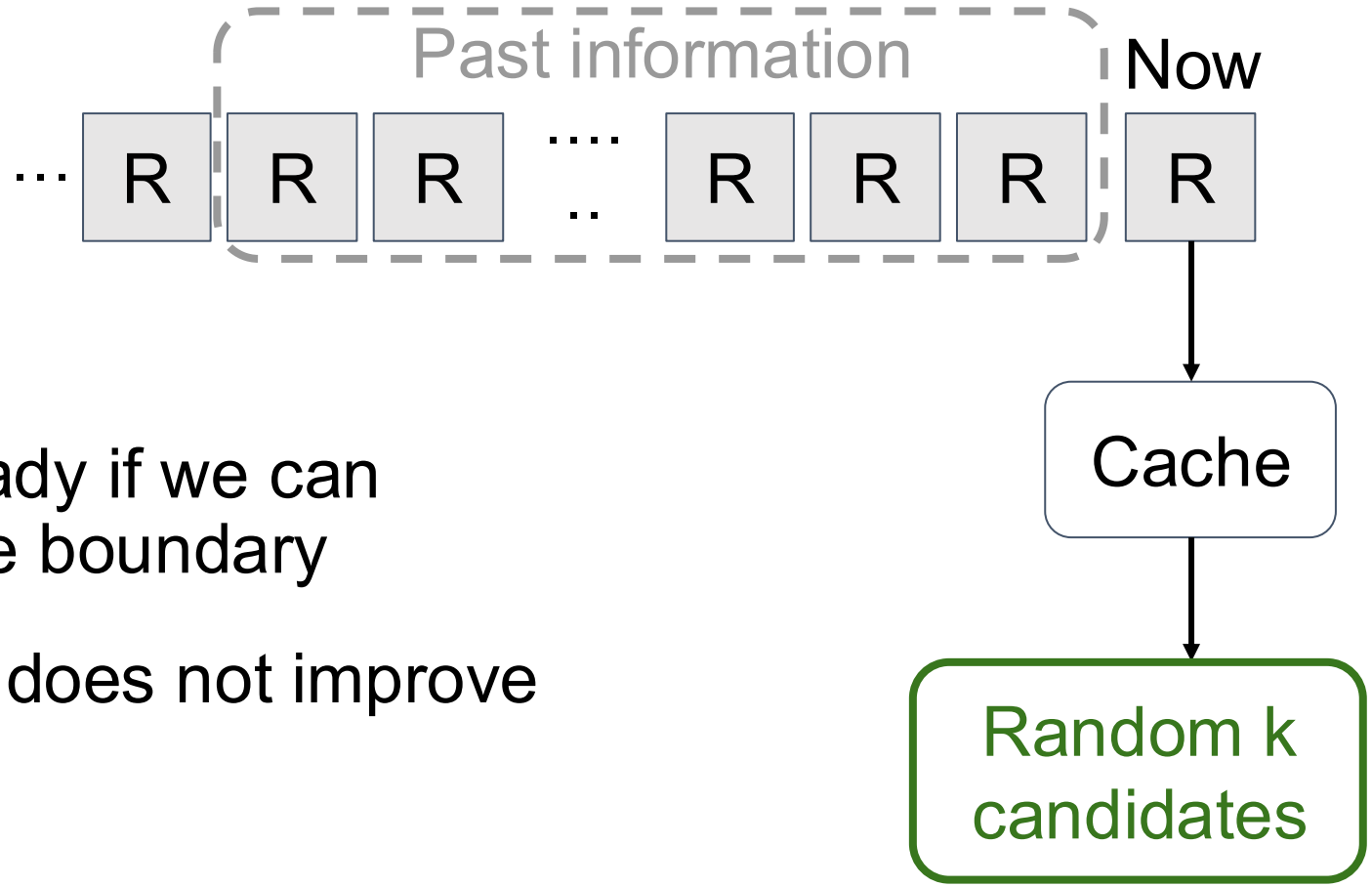
What past information to use?

Generate online training data?

What ML architecture to select?

How to select evict candidates?

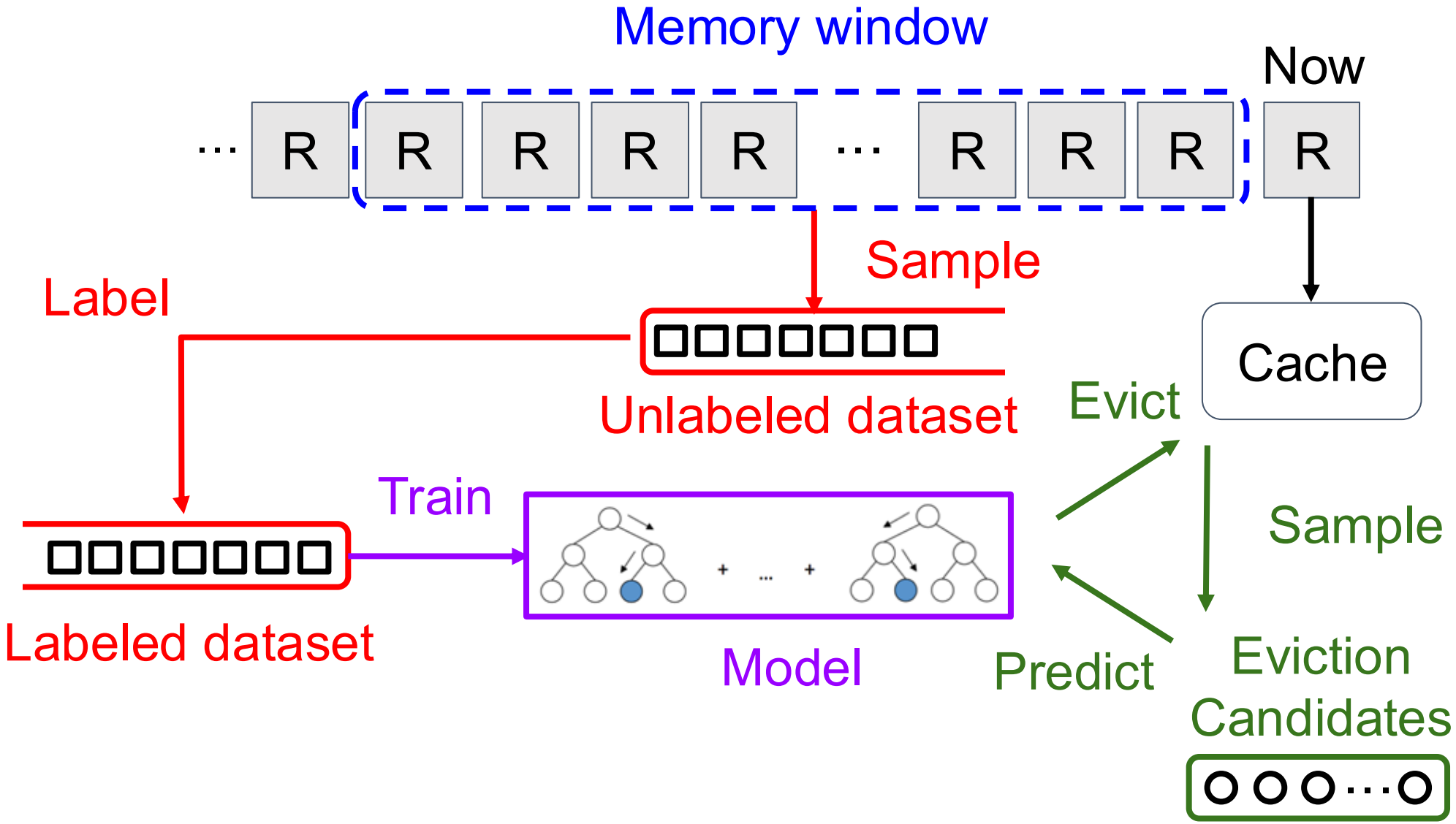
Solution 4: Random Sampling for Eviction



Can mimic relaxed Belady if we can find 1 object beyond the boundary

k=64 candidates; more does not improve good decision ratio

Learning Relaxed Belady



Implementation

- Simulator implementation
  - LRB + 14 other algorithms

- Prototype implementation
  - C++ on top of production system (Apache Traffic Server)
  - Many optimizations

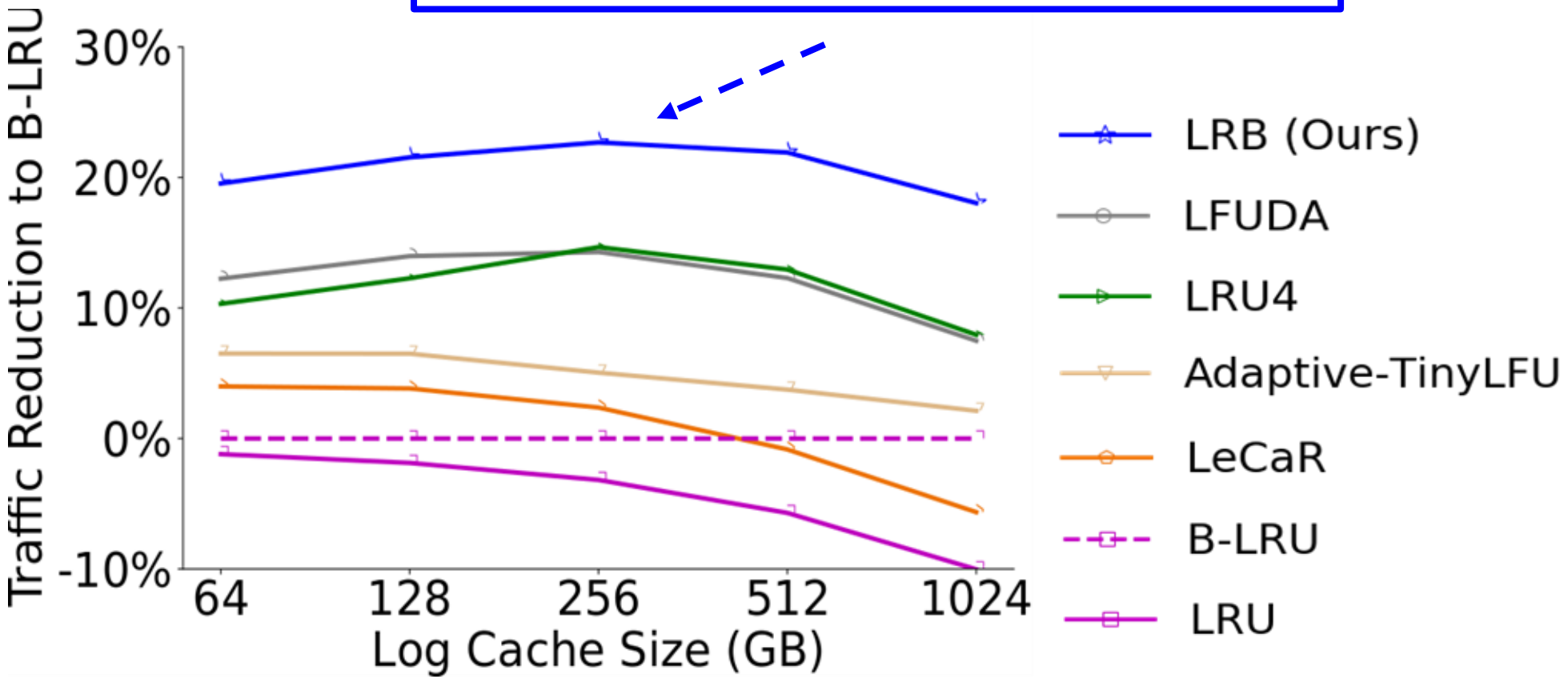
Evaluation Setup

- Q1: Learning Relaxed Belady (LRB) traffic reduction vs state-of-the-art
- Q2: overhead of LRB vs CDN production system
- Traces: 6 production traces from 3 CDNs
- Hyperparameter (memory window/model/...) tuned on 20% of trace

LRB Reduces WAN Traffic

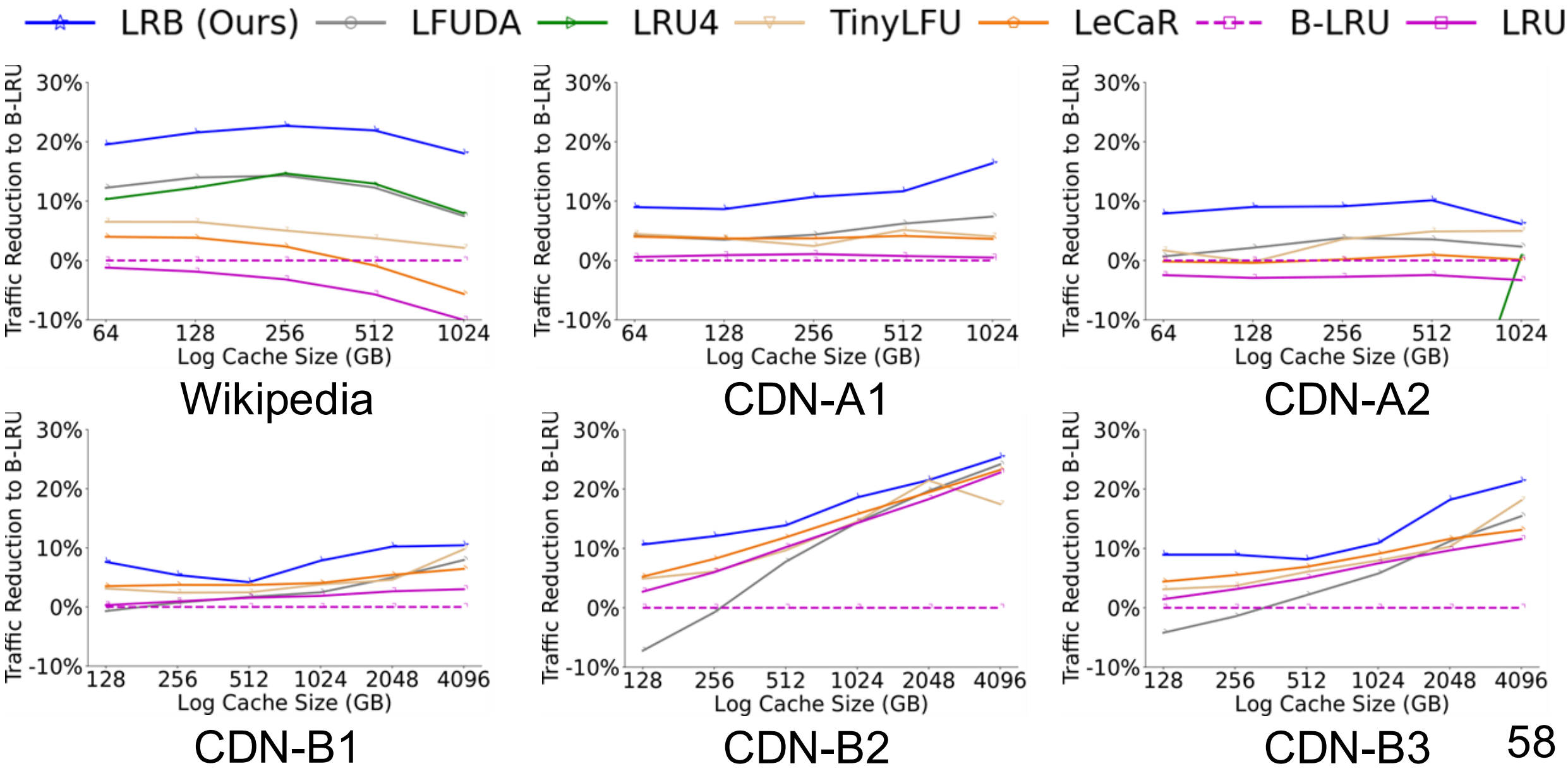
Industry standard

20% traffic reduction over B-LRU
10% reduction over the best SOA

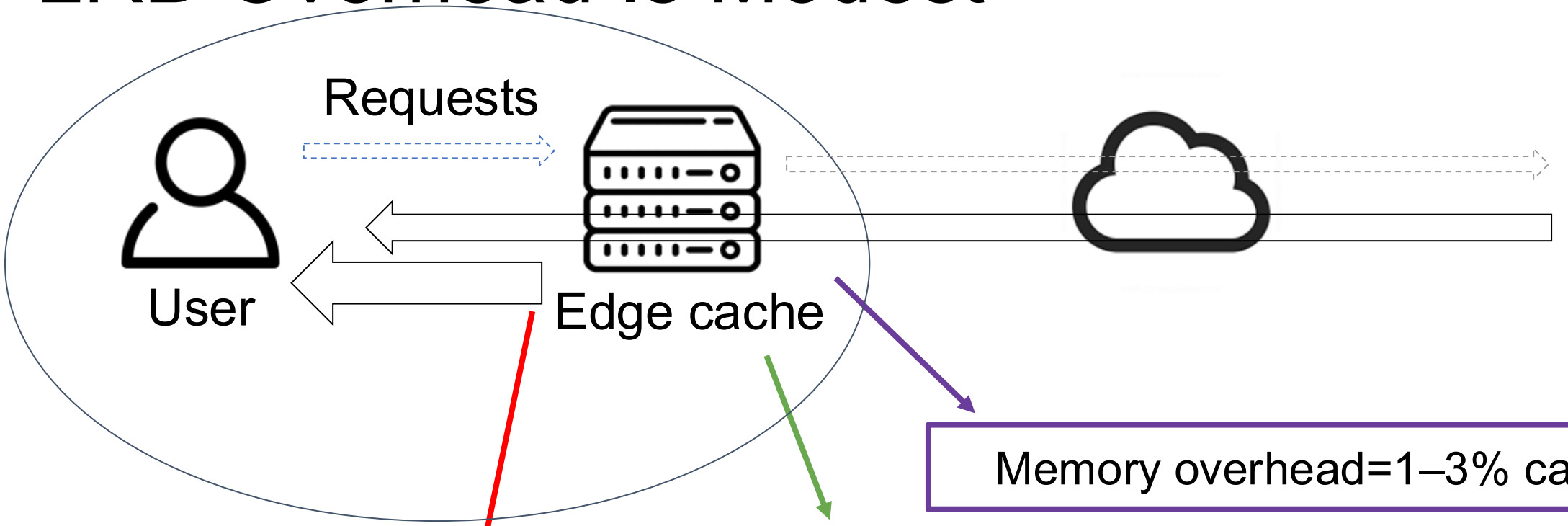


Wikipedia trace

LRB Consistently Improves on the State of the Art



LRB Overhead Is Modest



Memory overhead=1–3% cache size

Peak CPU: 16% vs 9% (unmodified)

Throughput: 11.7 Gbps vs 11.7 Gbps (unmodified)

Conclusion

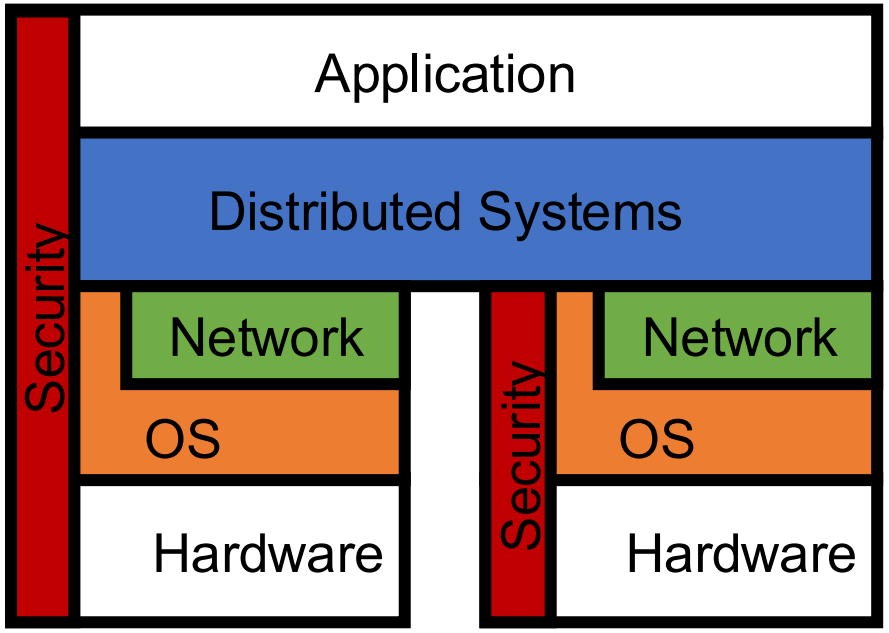
- LRB reduces WAN traffic with modest overhead
- ML-for-systems generally promising to replace heuristics
- Key insight: **relaxed Belady**

  →Simplifies machine learning & reduces system overhead

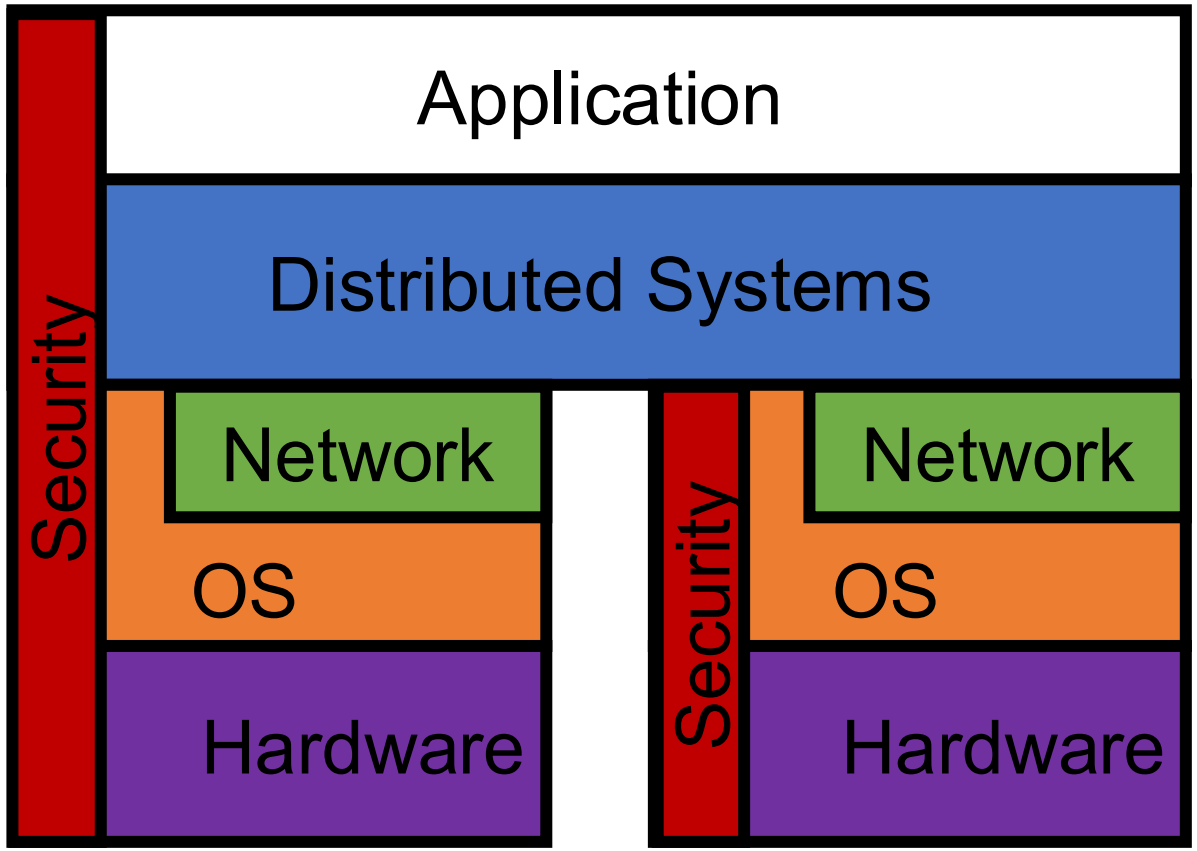

Microsoft Research

Systems!

- Systems abstract underlying resources
- Systems are **everywhere**
- Systems are challenging and interesting and cool
- This class was about systems

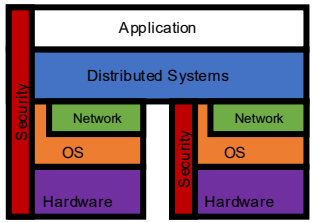


Systems You Can Learn More About



- Application
- Distributed Systems
- Networking
- Operating Systems
- Security
- Hardware

Systems You Can Learn More About



• Applications	COS 333 (Every semester)
• Distributed Systems	COS 418 (Fall ‘26)
• Networking	COS 461 (Spring ‘27)
• Operating Systems	COS 417 (Spring ‘27)
• Security	ECE/COS 432 (Spring ’27)
• Hardware (Processors)	COS/ECE 375 (Fall ‘26)

Remaining Time: Ask Me Anything!