Persistent Temporal Streams

David Hilley Umakishore Ramachandran

{*davidhi, rama*}*@cc.gatech.edu*

School of Computer Science
College of Computing, Georgia Institute of Technology

Middleware '09

Outline

- Intro
- Premise
- Temporal Streams
  - Programming Model
  - System Design & Implementation
  - Evaluation
- Conclusion & Questions

What is Live Stream Analysis?

- Live Stream Analysis
  - Surveillance / “Situational Awareness”
  - Traffic Analysis
  - Cargo / Asset Tracking
  - Robotics
  - Disaster Response
- Ubiquitous and increasingly important

What is Temporal Streams?

- Building blocks for stream analysis
- Distributed data structures for streams
- "Glue" for communicating components
- A lower-level substrate

Our Own Experience

- TV Watcher
- Media Broker / MB++
- Streaming Grid
- *V(A)aaS* – video-analytics-as-a-service
- RF^2ID
- ASAP – situational awareness
- IPTV Analytics / Recommender systems

Pain Points

- Time – synchronization, data retrieval
- Scalable data delivery
- Storage of streaming data
- Management of computation? – yes, but vastly different requirements between applications and domains

Pain Points

- Time – synchronization, data retrieval
- Scalable data delivery
- Storage of streaming data
- Management of computation? – yes, but vastly different requirements between applications and domains

Solution Space

- MPI, Message-Oriented-Middleware
  (too low level)
- Stream Data Management Systems
- Event Stream Processing (ESP) / CEP
  (too high level)
- Temporal Streams
  (just right)

Relationship to Other Work

- A middleground
  - Below full SDMS and stream languages
  - Above non-stream oriented distributed communications
- Roughly analogous to non-streaming:
  - Distributed Data Structures (Gribble, et al.)
  - BerkeleyDB (Seltzer, et al.)
  - Boxwood (MacCormick, et al.)

In Context

- Bottom-up “systems” approach
- Recognizing the value of time
- Explore effect on system level properties
- Programming model incorporates time
- Unique point in the design space

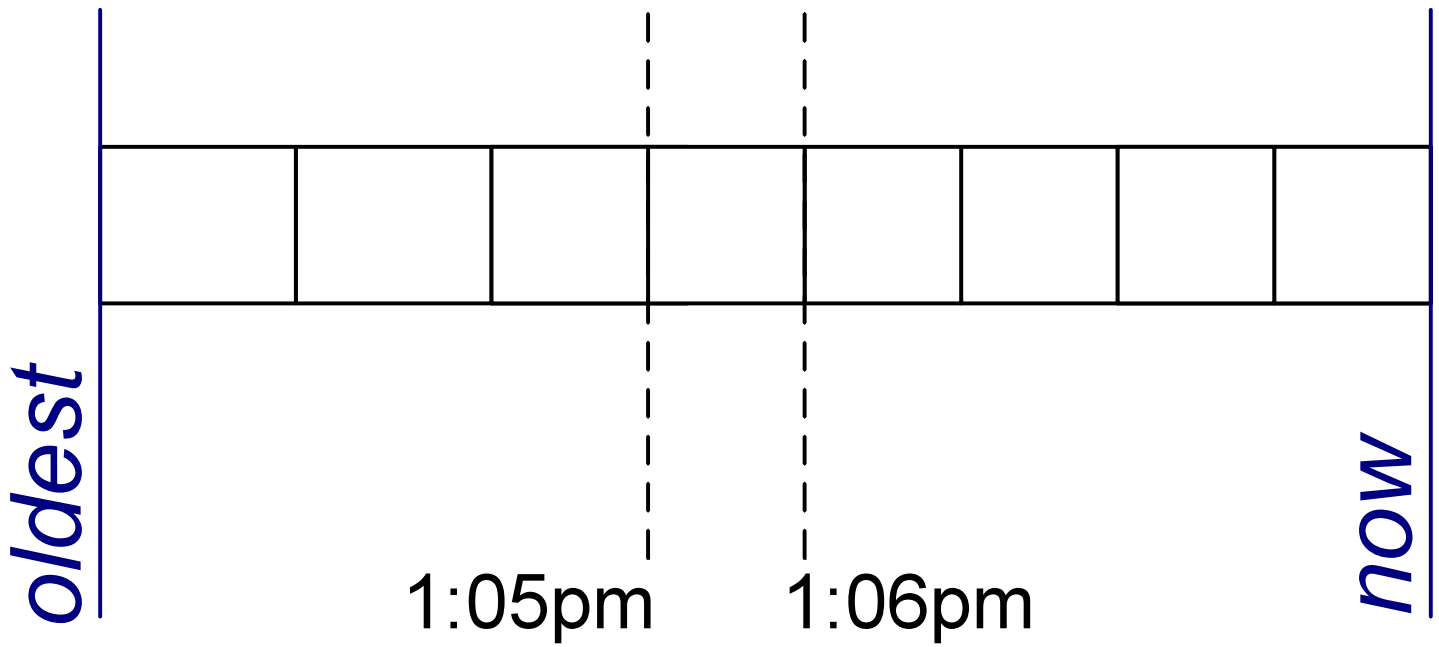
Going forward

- Intro
- Premise
- Temporal Streams
  - Programming Model
  - System Design & Implementation
  - Evaluation
- Conclusion & Questions

Programming Model

Channel

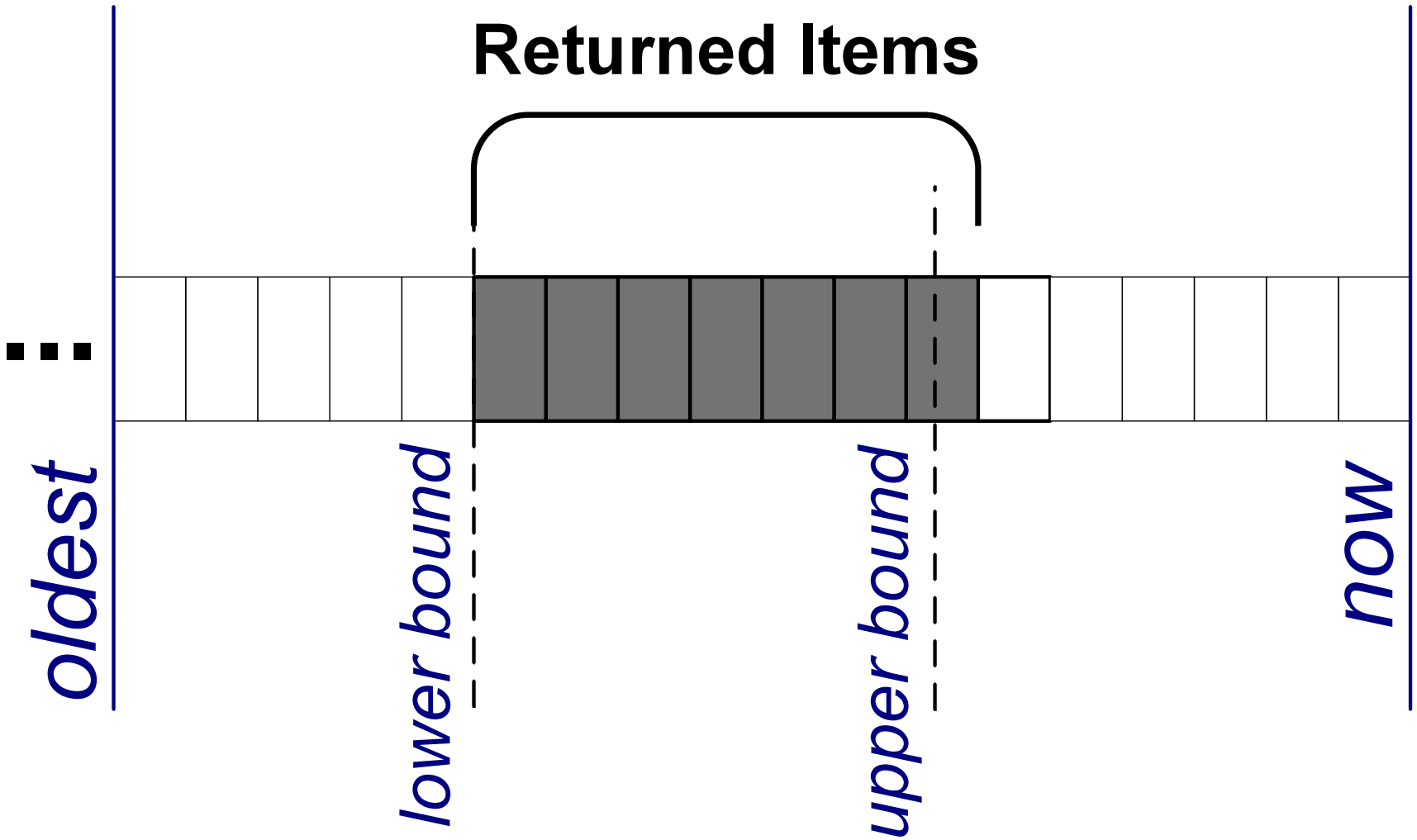
- Represents a continuous data stream
- Items ordered by wall-clock timestamp



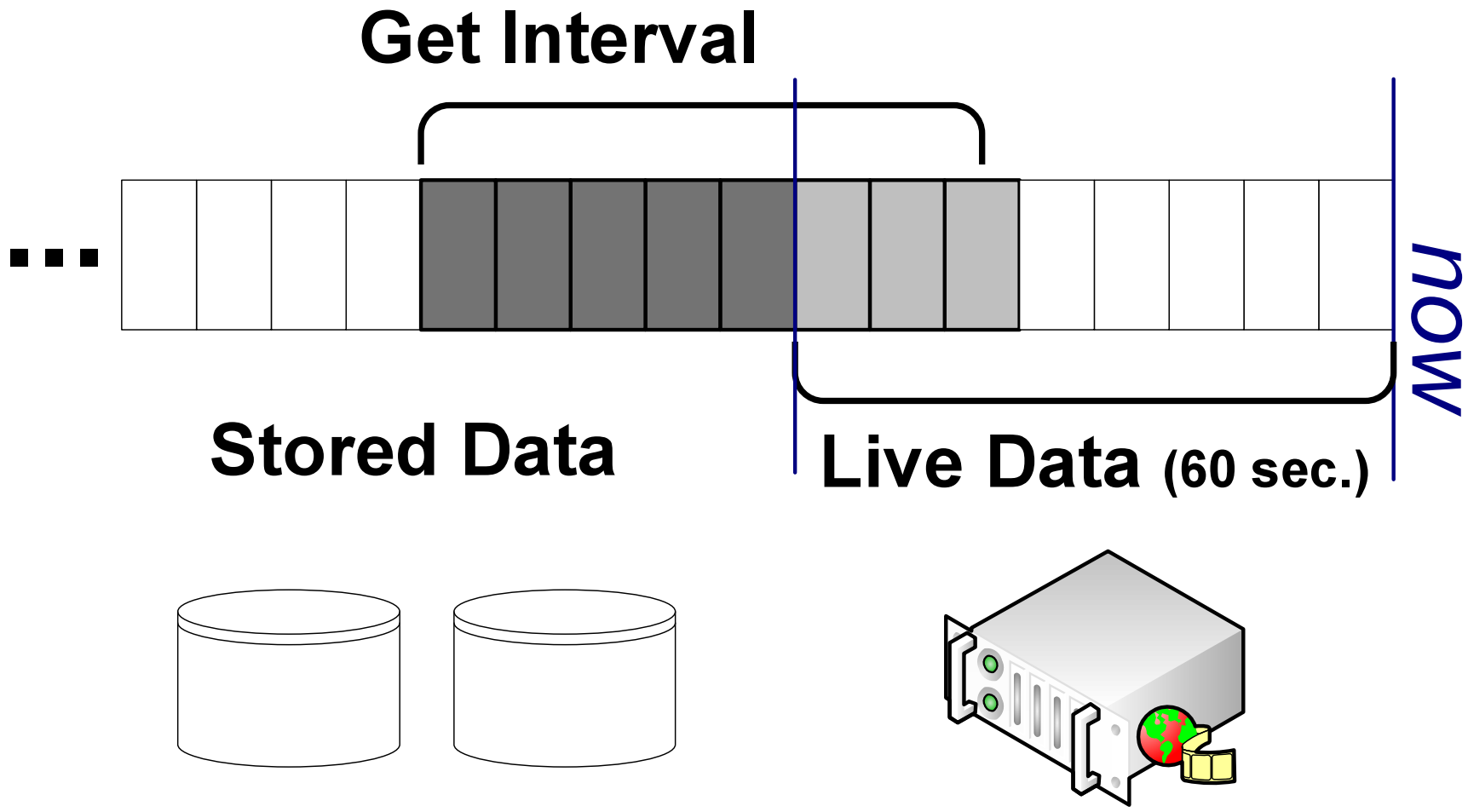
Channel

- Represents a continuous data stream
- Items ordered by wall-clock timestamp
- Simple time-based operations:
  - *put*(item, [timestamp])
  - *get*(lower_bound, upper_bound)
- Time variables to specify time intervals
  e.g. *now*, *newest-after*(`ts`), *oldest*, etc.
- Spans communication & storage

Channel Get Interval Example



Persistent Channel Get



Stream Persistence

- Seamless persistence with same interface
- System automatically manages:
  - moving "live" items to backing store
  - retrieving stored items when necessary
- Control storage representation:
  - User-provided transformation (pickling)
  - Automatic adaptation
- Pluggable storage backends

The Premise

- Not just about "plumbing" / transport
- More explicit support for writing stream manipulation code via the data abstractions
- Wall-clock time as a recognized entity
- Time as an indexing mechanism naturally admits synchronization, data persistence

As we continue

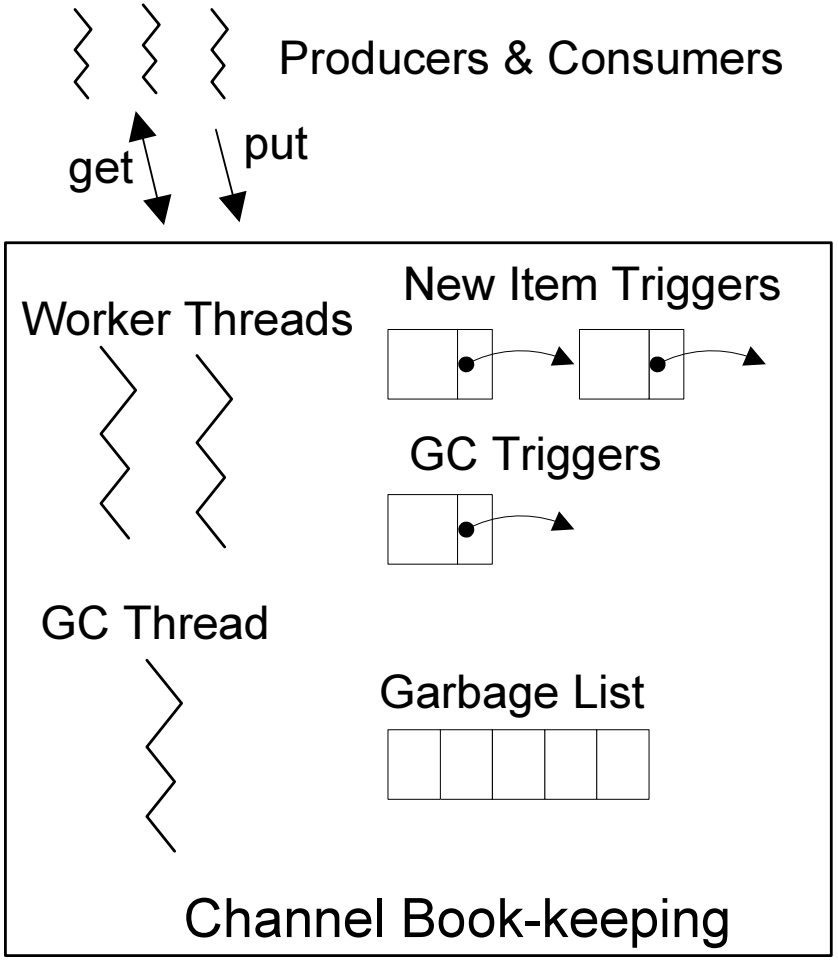
- Intro
- Premise
- Temporal Streams
  - Programming Model
  - System Design & Implementation
  - Evaluation
- Conclusion & Questions

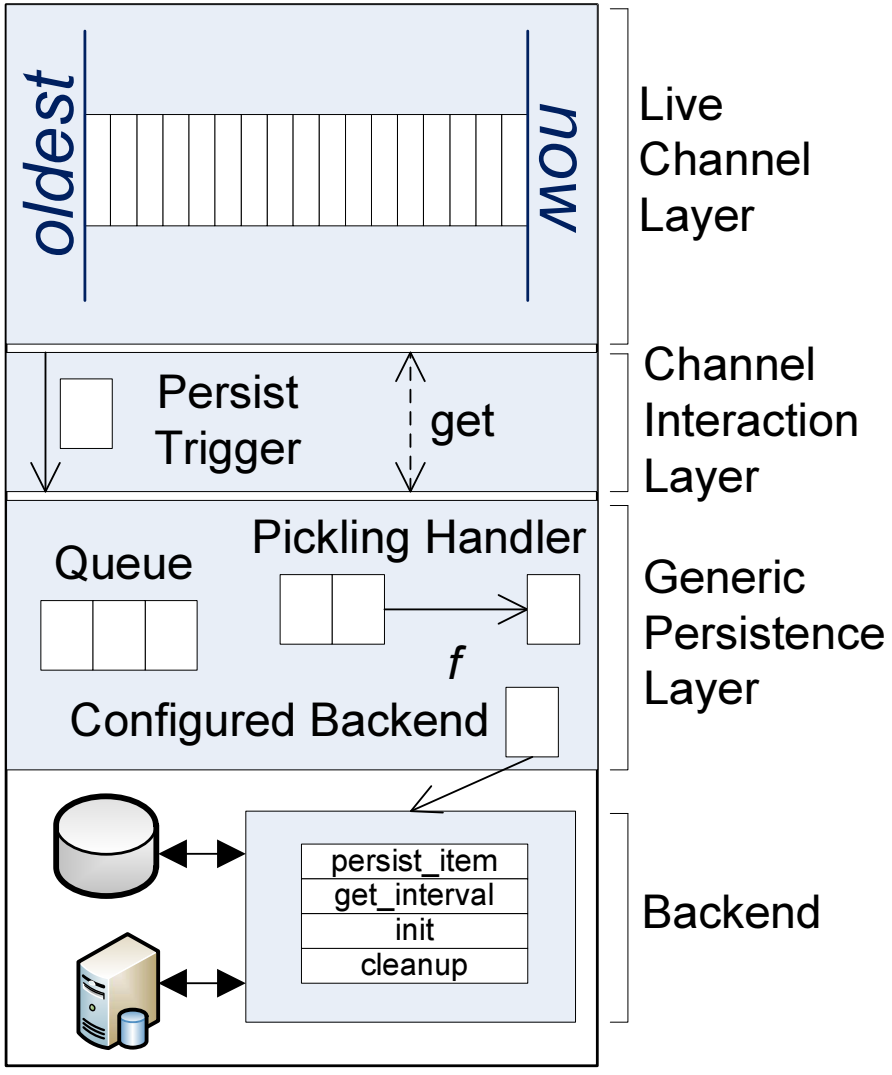
System Design & Implementation

Channel Data Management

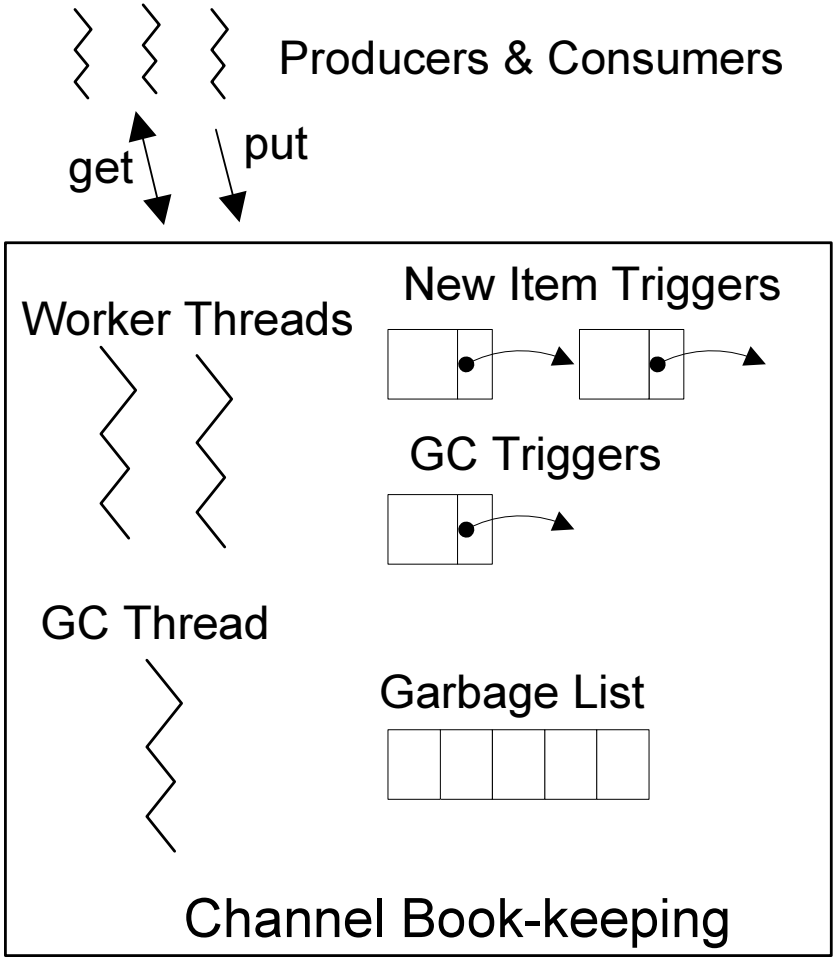
- Significant portion of system complexity
- Index live and stored data
- Interface with storage backend
- Pickling – content transformation
- Data delivery
- Connection management
- Lifecycle management

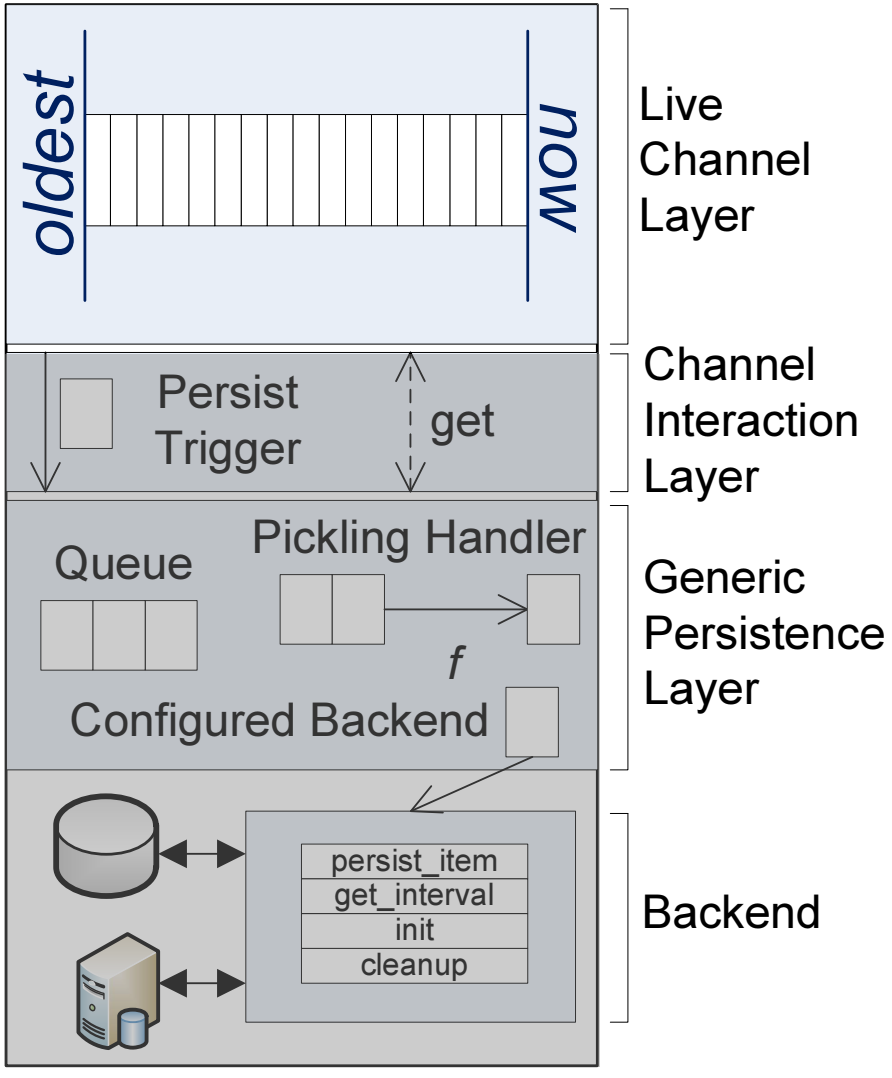
Channel Architecture





Channel Architecture





Three Layer Stack

- Channel interaction layer
  - Hooks in live channel
  - Sends data to generic persistence layer
- Generic persistence layer
  - Queues items for transformation
  - Guarantees temporal ordering and one writer
- Backend layer
  - `init`, `get_interval`, `put_item`
  - Interfaces with concrete storage

Filesystem-based Backend

- Exploit properties of the above layer
- Like ISAM, but append-only
- Concurrent reads & one writer, no locks
- Treat secondary storage like a big ring-buffer
- Based on results from Hyperion/StreamFS (USENIX '07)

Moving along

- Intro
- Premise
- Temporal Streams
  - Programming Model
  - System Design & Implementation
  - Evaluation
- Conclusion & Questions

Evaluation

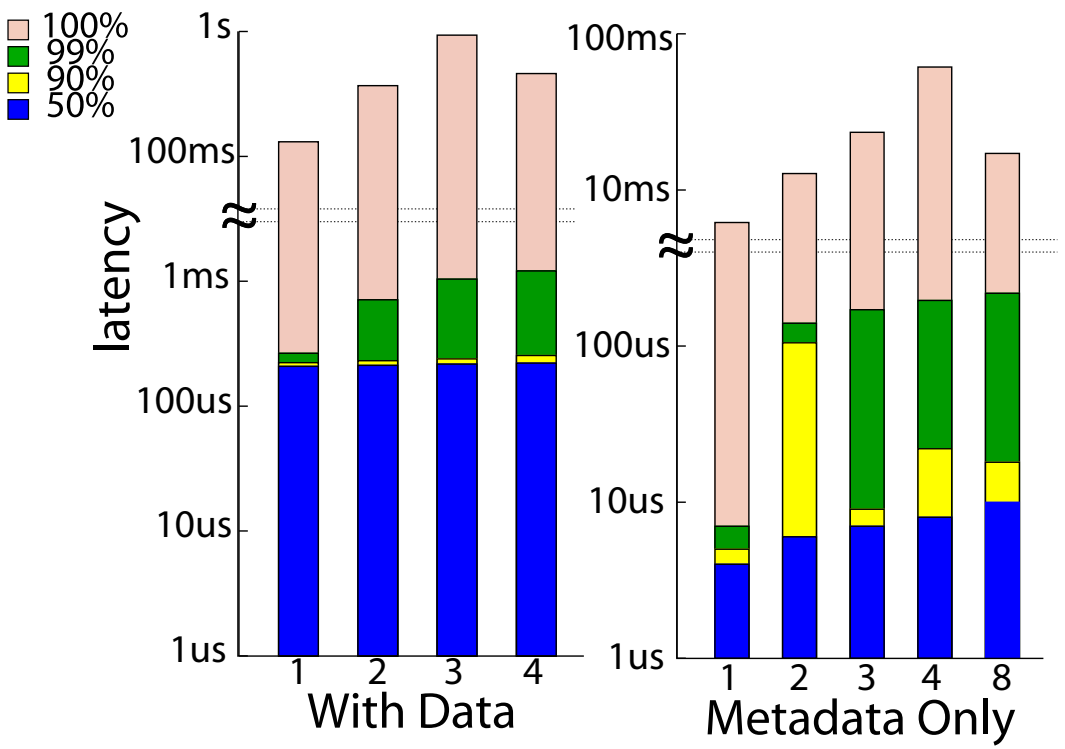
Evaluations

- System-level evaluation:
  - Quantitative overhead of components
  - Storage & retrieval
  - Scaling under contention
  - Pickling
- Application-based evaluation:
  - Airport surveillance
  - Video analysis & storage
  - Historical queries
- Show you can achieve good performance

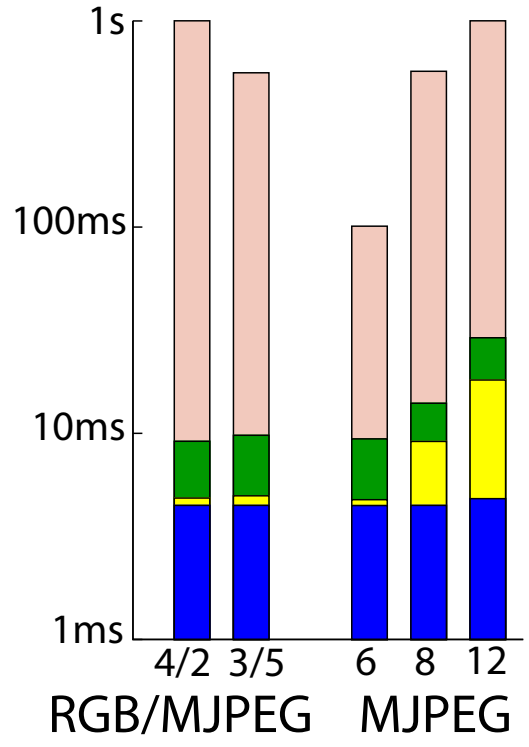
Channel Persistence Evaluation

- Single consumer backend overhead
  - OProfile system-wide measurements
  - Callgrind instructions in backend
- Single consumer baseline get overhead
- Multiple stream scaling
  - Metadata baseline
  - RGB video data
  - RGB with pickling handlers
- Dynamic load adjustment w/ pickling
- Mixed live/stored workload

Multiple Stream Scaling



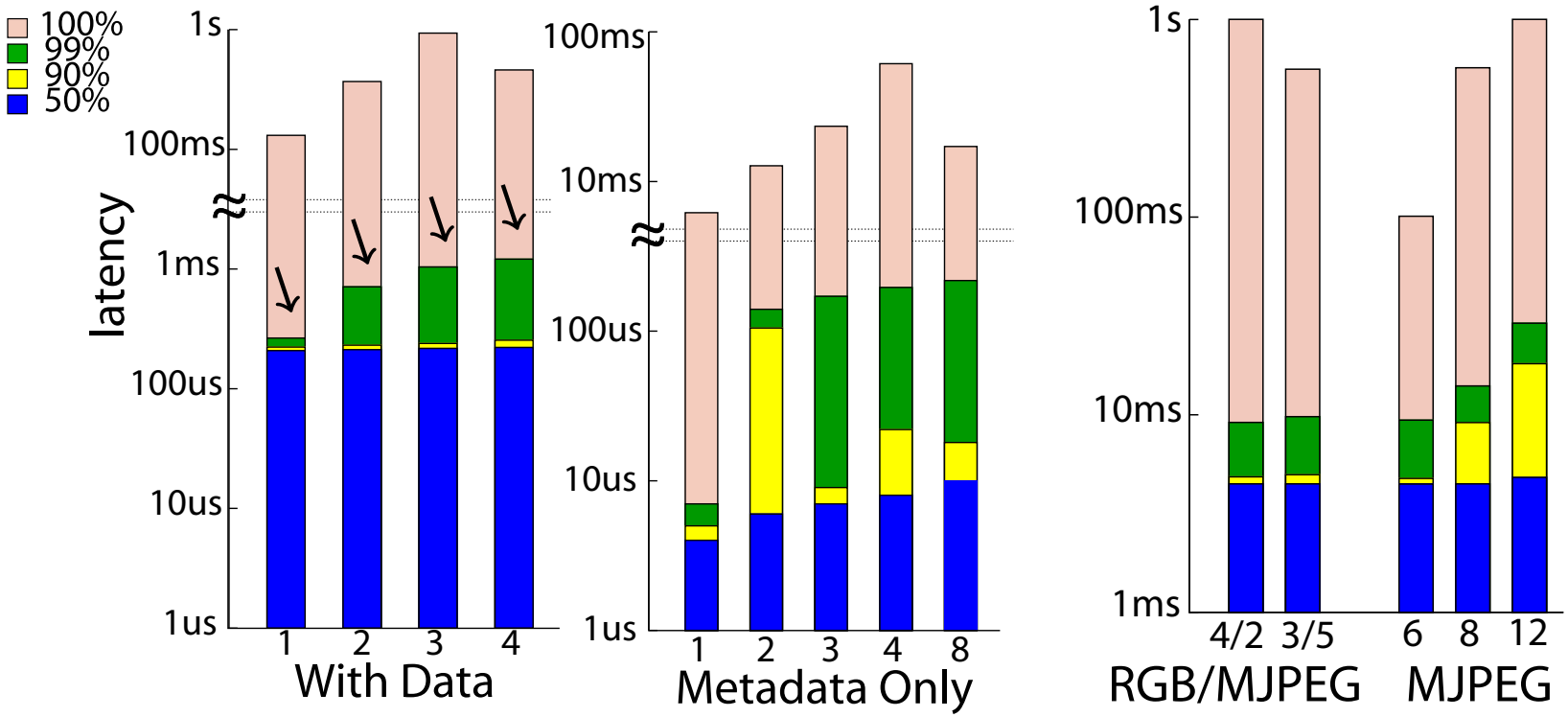
(a) RGB video streams



(b) with JPEG pickling

Item latencies by statistical percentile

Multiple Stream Scaling

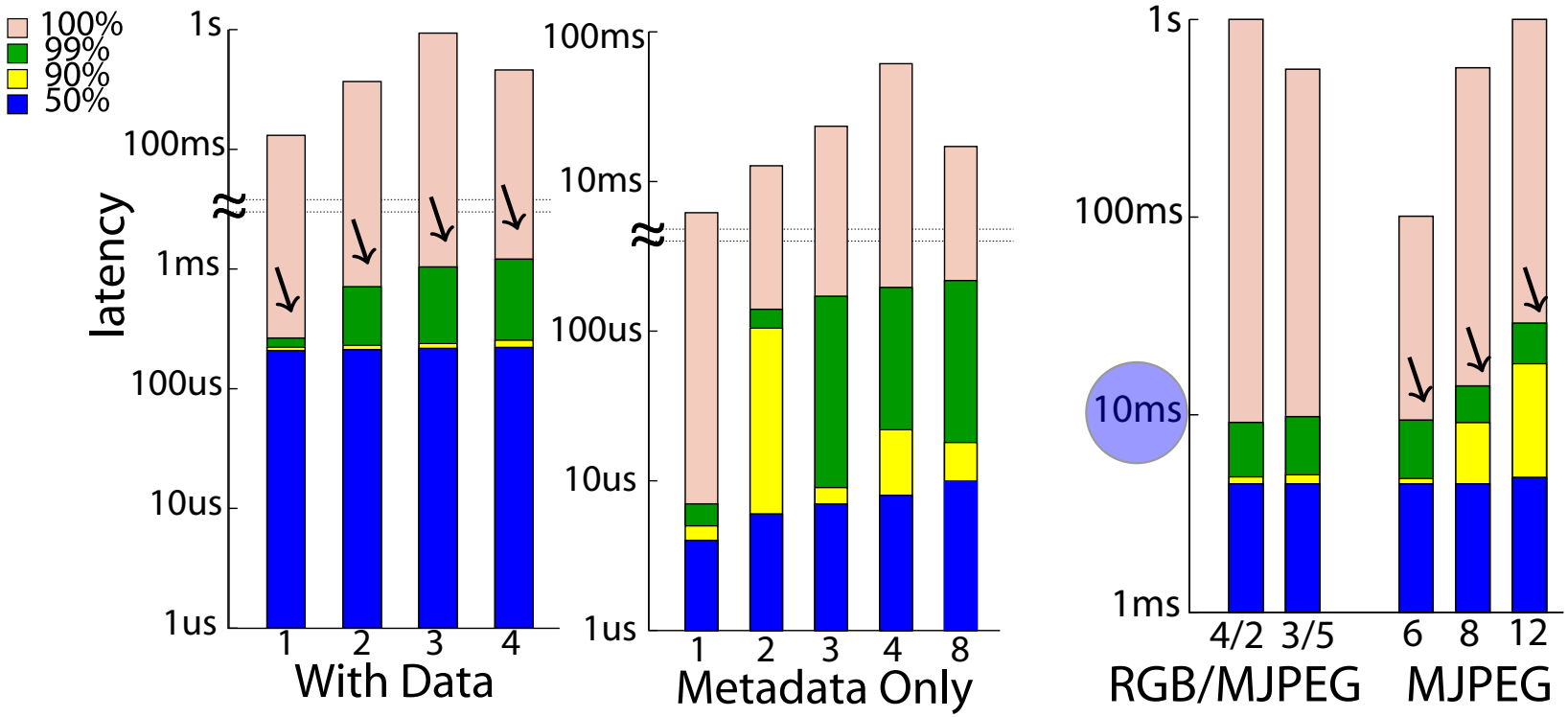


(a) RGB video streams

(b) with JPEG pickling

Graceful scaling under increasing contention

Multiple Stream Scaling



(a) RGB video streams

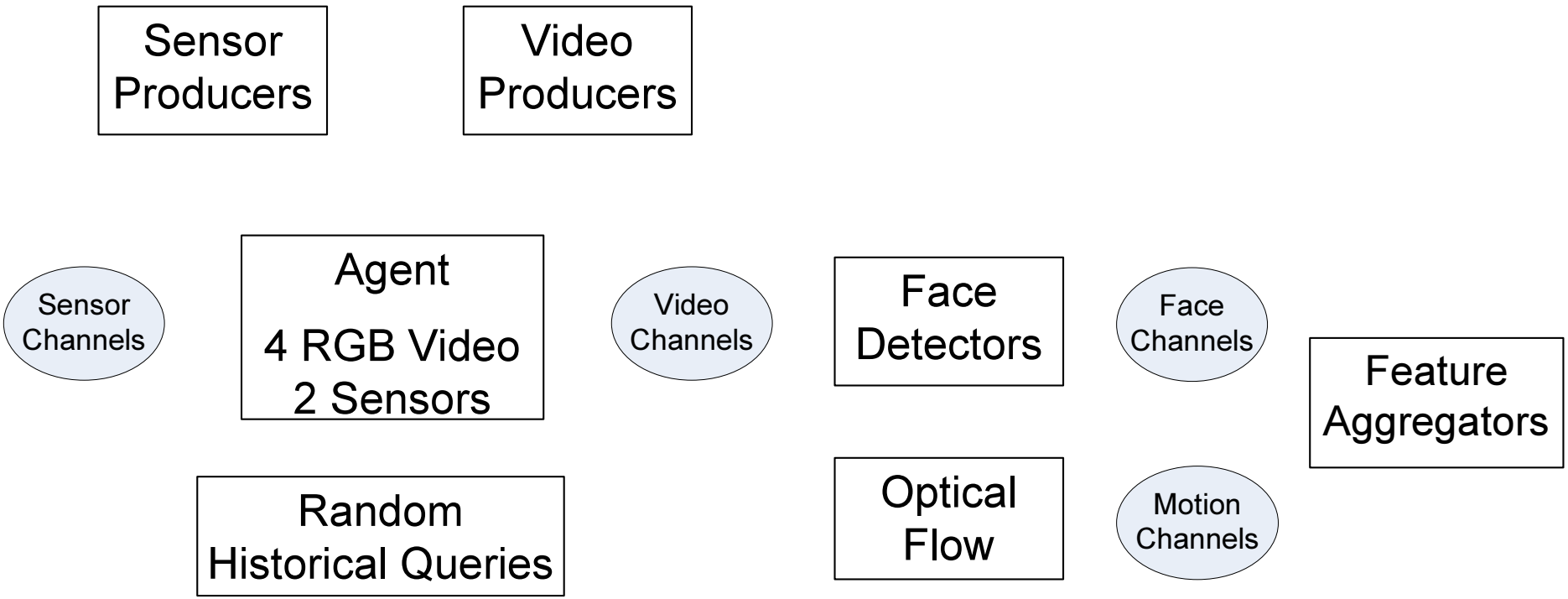
(b) with JPEG pickling

Graceful scaling under increasing contention

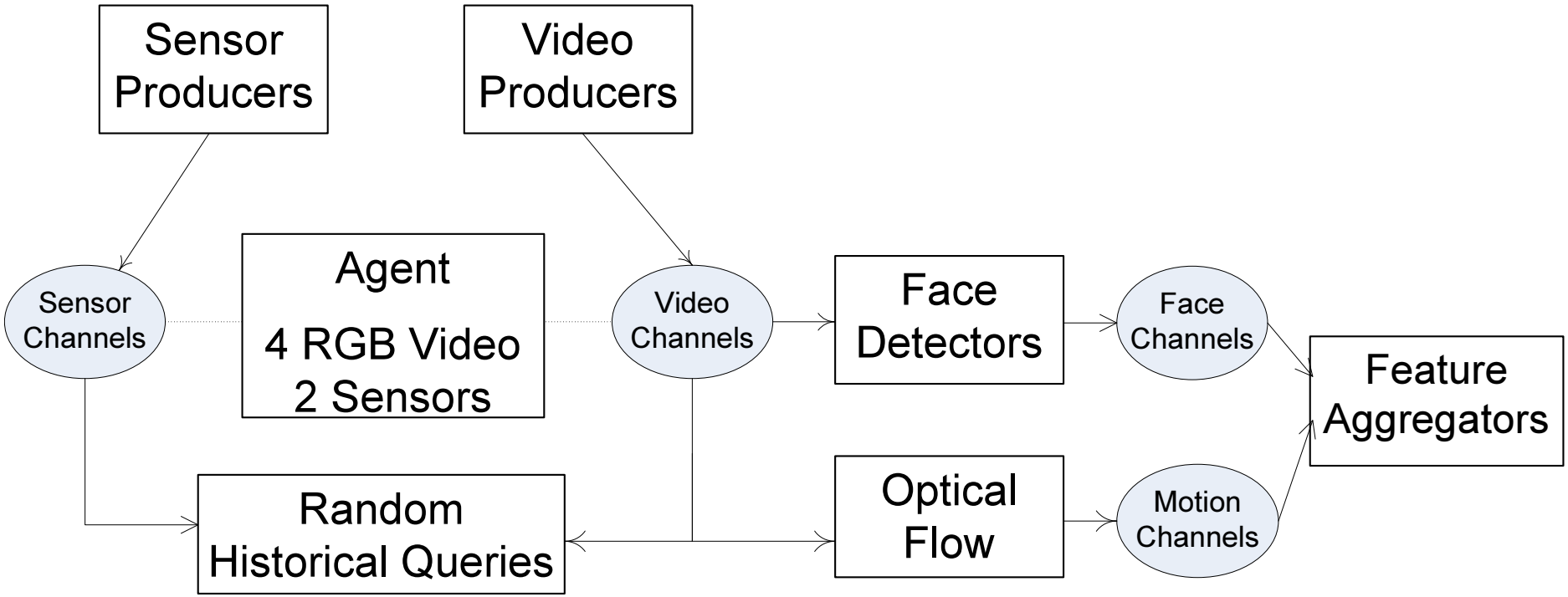
Airport Surveillance

- Application
  - Pick streams of interest
  - Store historical data
  - Approximate interest
  - Modeled on ASAP

Components & Dataflow



Components & Dataflow



Measurements

- TS Historical Queries: 4-15ms
- ASAP Published Live Queries: 135-175ms

Analysis?

- Same hardware, same feature detectors.
- ASAP's comms layer reinvents time-oriented access anyway.
- Temporal streams gives you stream persistence.
- Temporal streams also allows transparent overlapping of communication and computation!

Recap

Value of time at the system level.

- Time-based distributed data structures
- Automatic data management
- Transparent stream persistence

Bottom-up approach.

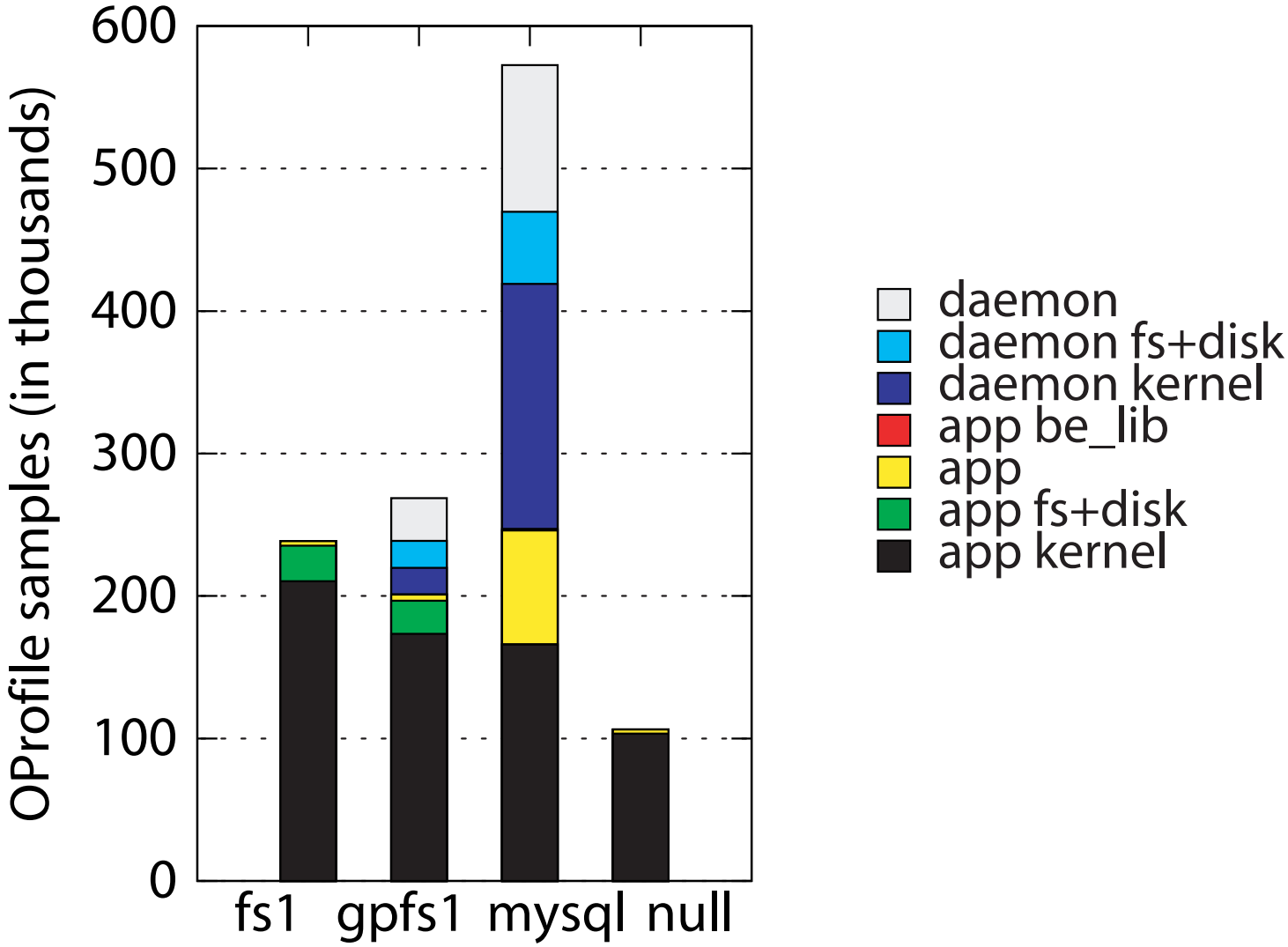
- No computational model imposed
- Lightweight; loose coupling
- Stream *data* substrate

That's all folks

- Thank you!
- Questions?

Extra Slides

Backend Comparison

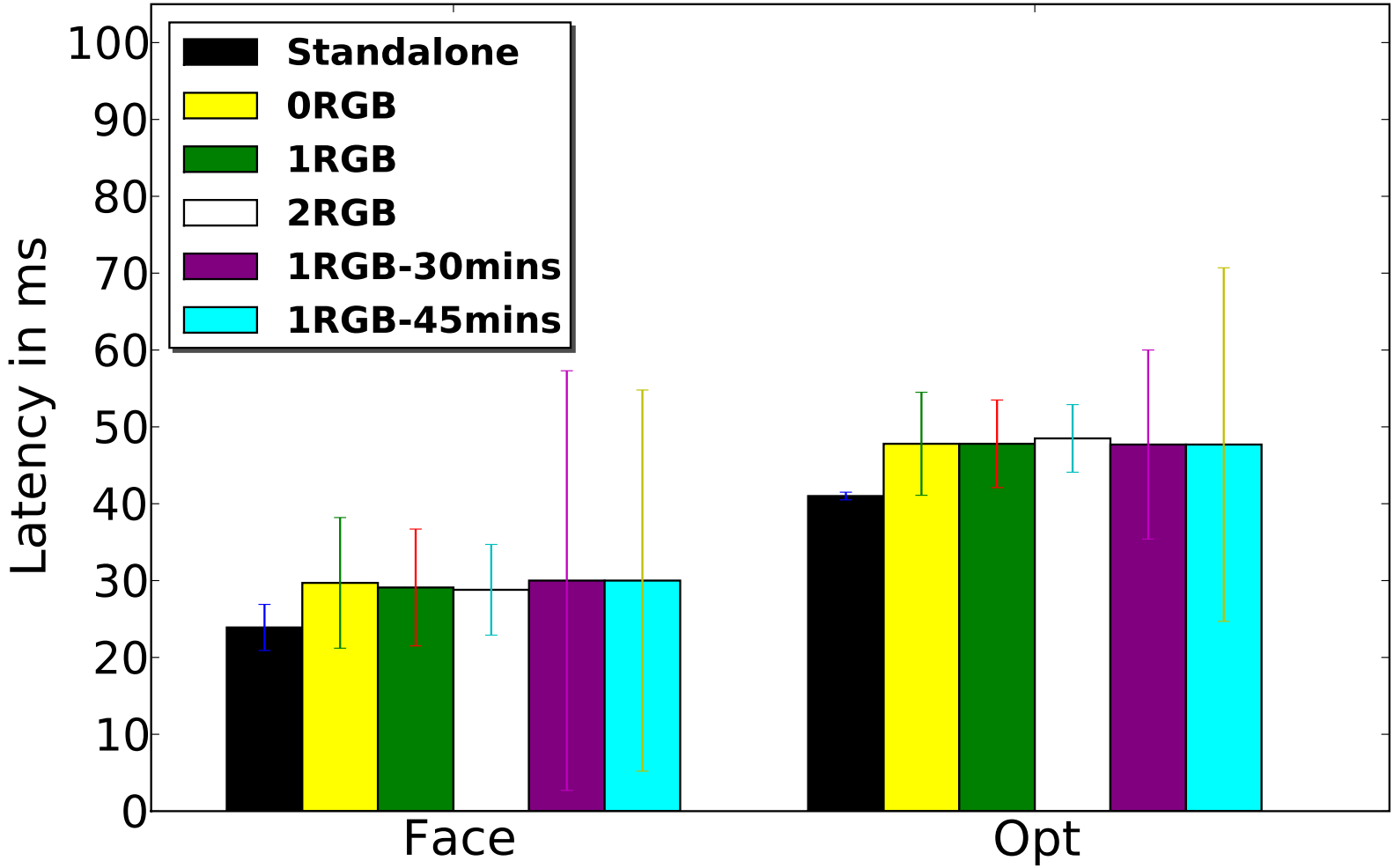


OProfile data: same workload, different backends.

Measurements

- Standalone feature detectors
- “In system” latencies
- Pipeline latencies
- Effect as RGB streams are added
- Effect as working set grows
- Historical data queries
- Data overheads for protocol versions

Measurements



15-25% includes network-related overhead and contention

Related Work

Categories

- Parallel Batch Processing
- Stream Databases / Processing Engines
- Stream Languages
- Distributed Programming
- Processing Streams/FIFOs

Relationship to Other Work

- A middleground
  - Below full SDMS and stream languages
  - Above non-stream oriented distributed communications
- Roughly analogous to non-streaming:
  - Distributed Data Structures (Gribble, et al.)
  - BerkeleyDB (Seltzer, et al.)
  - Boxwood (MacCormick, et al.)

Key Differentiation

- No computational model imposed
- Lightweight; loose coupling
- Stream *data* substrate

One Example

Unblinking Eyes, for $20 Million, at Freedom Tower
September 24, 2008

"If you have ever wondered how security guards can possibly keep an unfailingly vigilant watch on every single one of dozens of television monitors, each depicting a different scene, the answer seems to be (as you suspected): they can't.

Instead, they can now rely on computers to constantly analyze the patterns, sizes, speeds, angles and motion picked up by the camera and determine - based on how they have been programmed - whether this constitutes a possible threat. In which case, the computer alerts the security guard whose own eyes may have been momentarily diverted. Or shut."

The New York Times, City Room blog

Common Traits

- Continuous streams
  - Heavy-weight
  - Unstructured
- Feature detection/extraction
  - Computationally intensive
  - Distributed analysis
- Historical data

Current Approaches

- Centrally controlled & managed execution
  - Stream Data Management System (SDMS)
  - Stream Processing Engines
  - Declarative query languages
  - Continuous queries (CQ)
- Independent communicating components
  - General distributed programming
  - Loose coupling
  - Lower level / more procedural

Gap

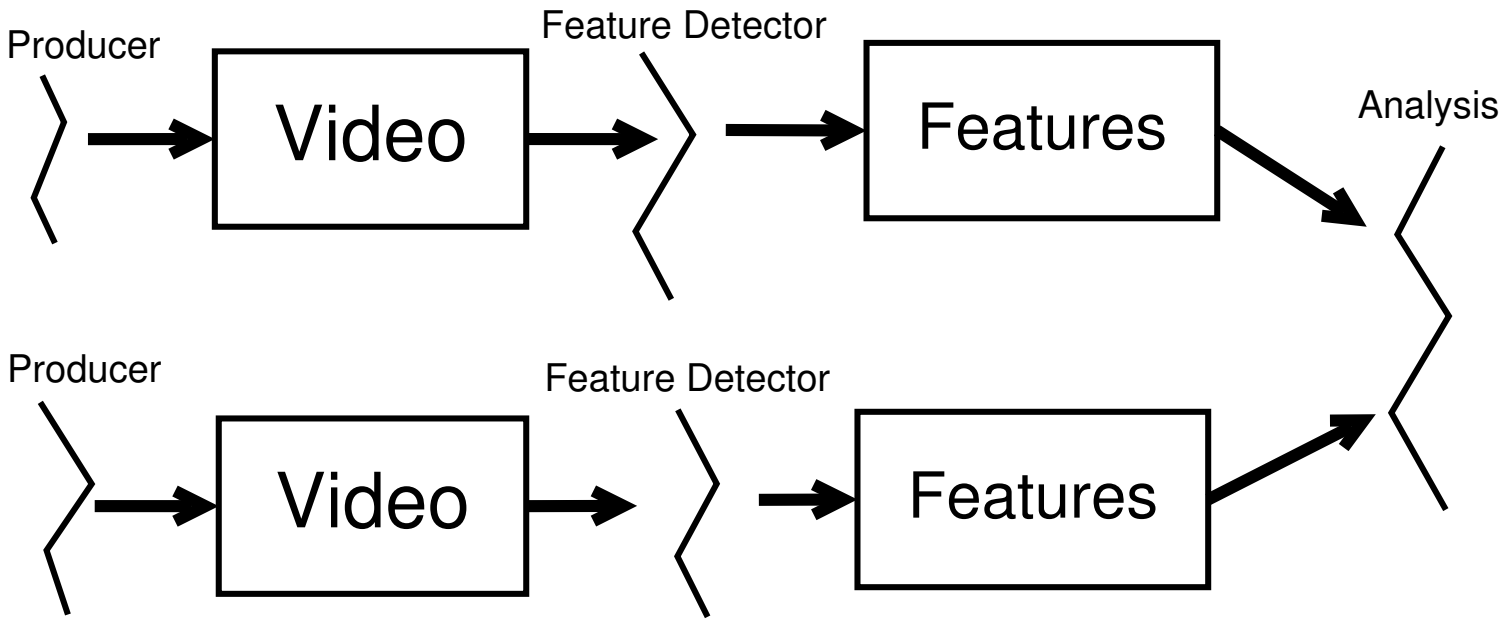
- Integrate:
  - Higher-level time-based model
  - General distributed application approach
- Keep looser coupling
- Independent communicating components
- Distributed data structures

Relationship to Other Work

- A middleground
  - Below full SDMS
  - Above non-stream oriented distributed communications
- Roughly analogous to non-streaming:
  - Distributed Data Structures (Gribble, et al.)
  - BerkeleyDB (Seltzer, et al.)
  - Boxwood (MacCormick, et al.)

Programming Model

- Distributed data structures – *channel*
- Threads communicate via channels



Time Variables

Variable	Param(s)	Definition
now	-	current time
newest	`ch`	timestamp of newest visible item in channel `ch`
oldest	`ch`	timestamp of oldest visible item in channel `ch`
newest-after	`ch`, `ts`	timestamp of newest visible item in channel `ch` if the item's timestamp is after `ts`, ∞ otherwise
next	`ts`	`ts` $+ \epsilon$ (the smallest timestamp $>$ `ts`)

Producer Code Example

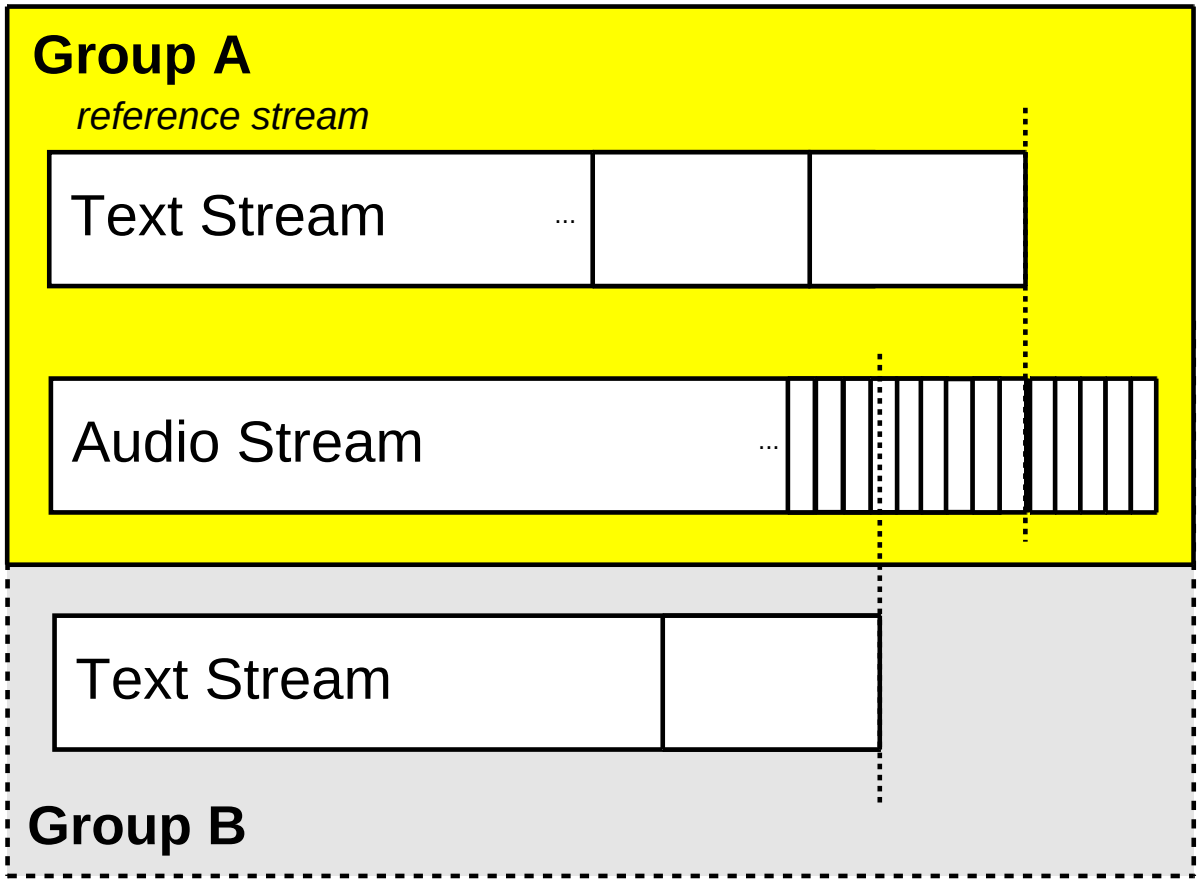
```
while(true) {
   Item item = new Item(dataBuffer);

   // Put the item into the channel with
   // the default timestamp of 'now'
   chanConn.putItem(item);

   ...
}
```

Consumer Code Example

```
Time32 lower = Time32.newest;
Time32 upper = Time32.v_now;

while(true) {
  Item item = chanConn.GETITEM(lower, upper);

  // Do something with the item
  ...

  lower = Time32.NEXT(item.getTimestamp());
}
```

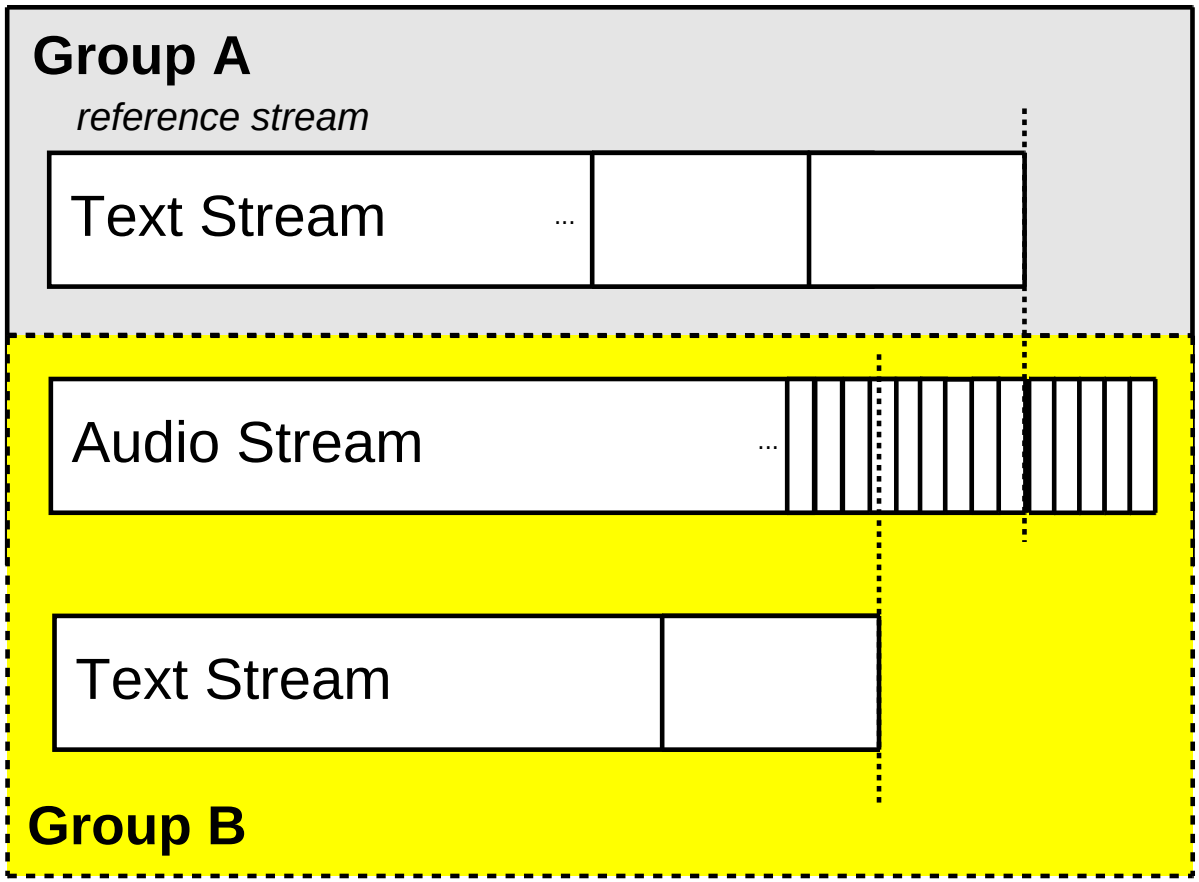
Channel Groups

- Only modify the visibility of items
- One channel may belong to many groups



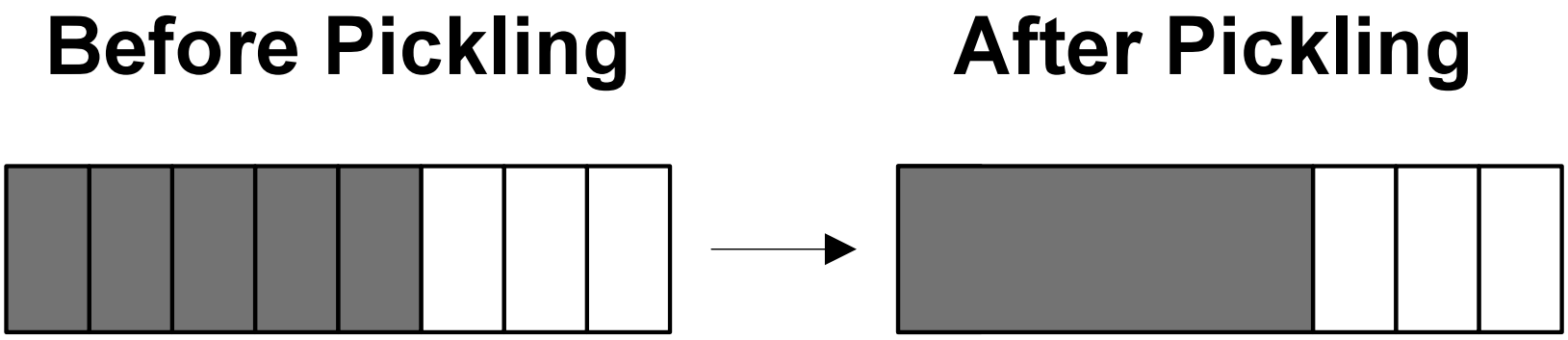
Channel Groups

- Only modify the visibility of items
- One channel may belong to many groups

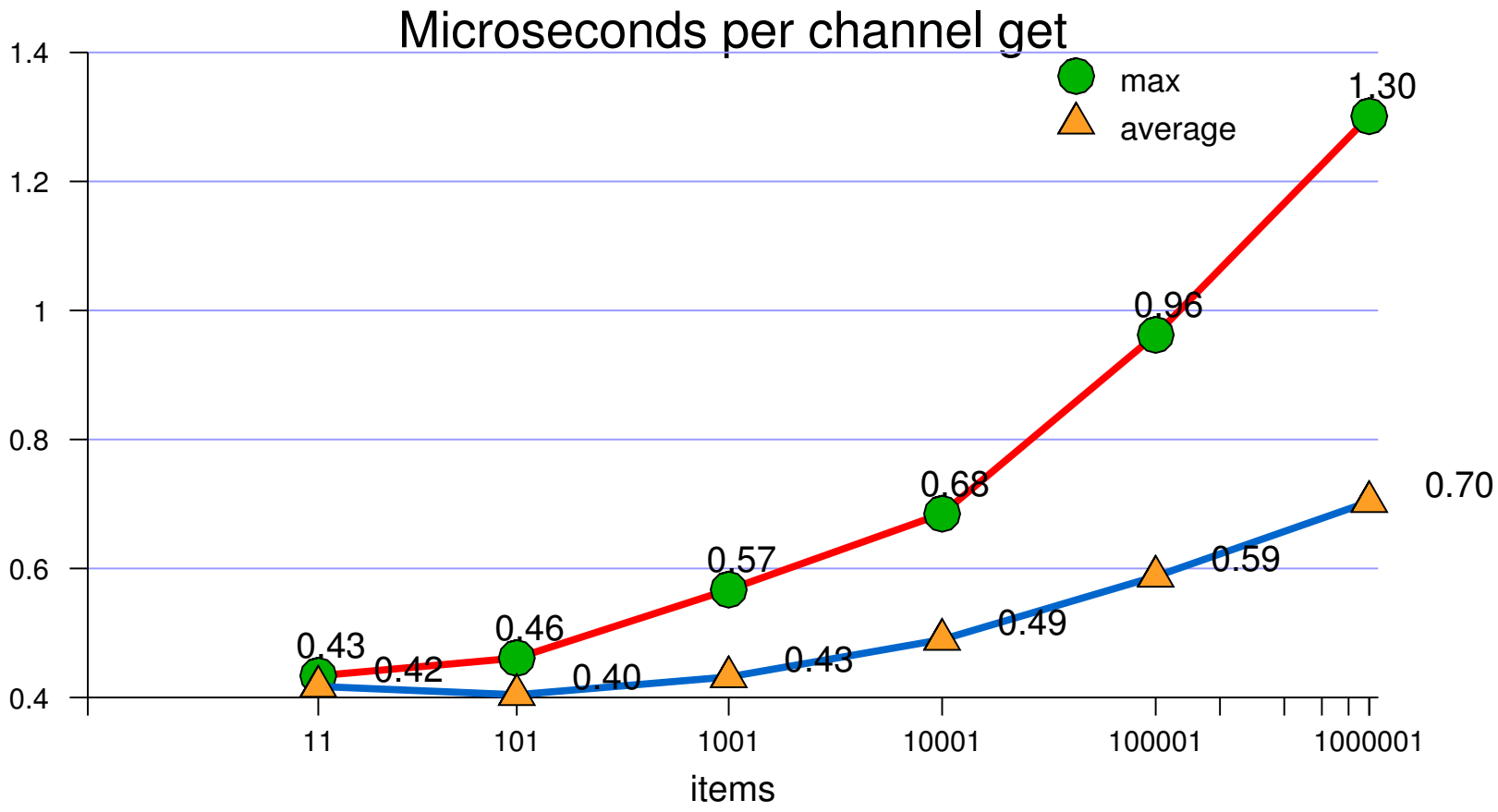


Persistent Representation

- Application provided *pickling handler*
- Arbitrary N-to-1 mapping function
- Transforms item for storage
- Vary pickling function by load
- Preserves time interval correspondence

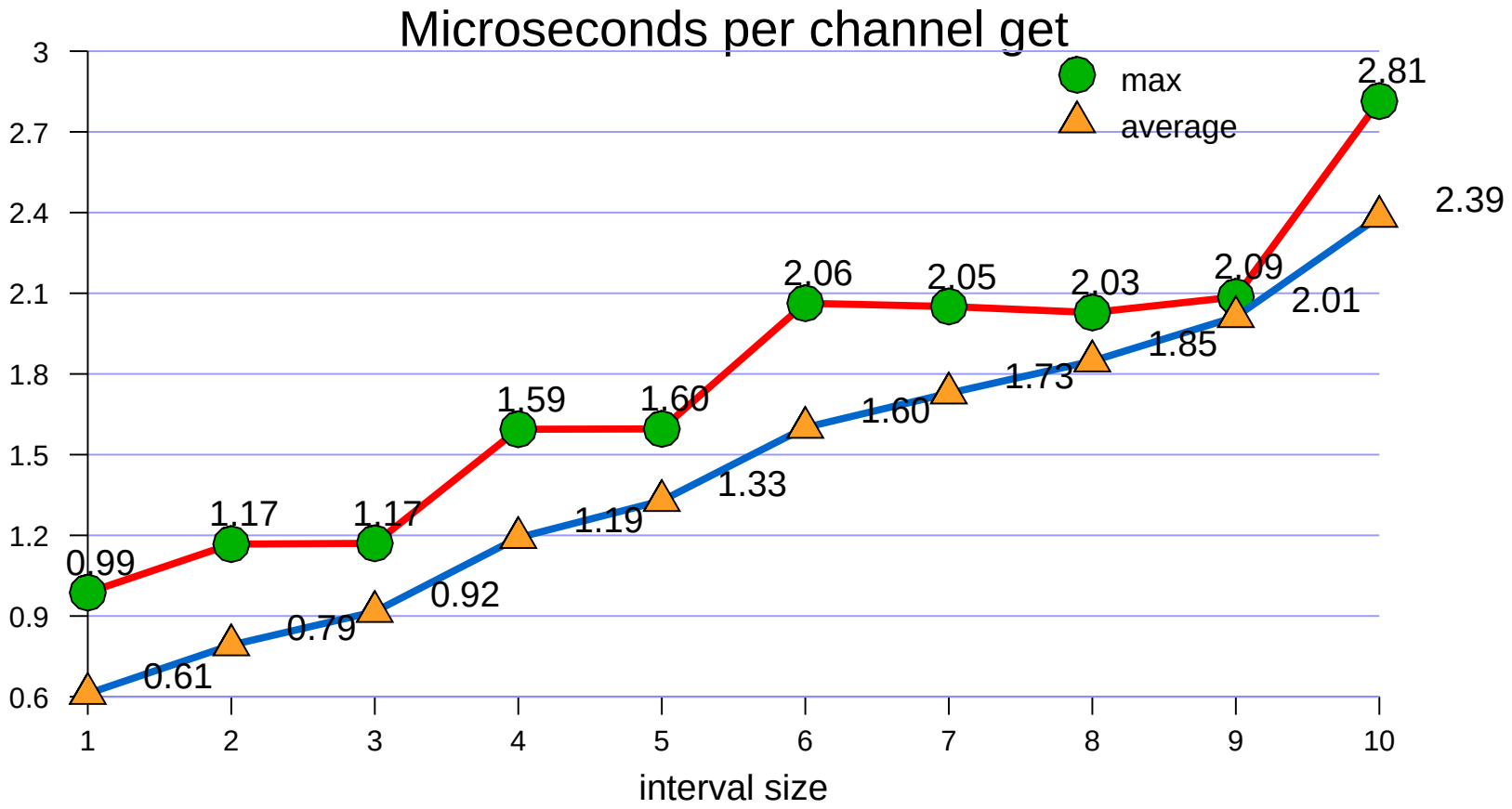


Local Channel Gets



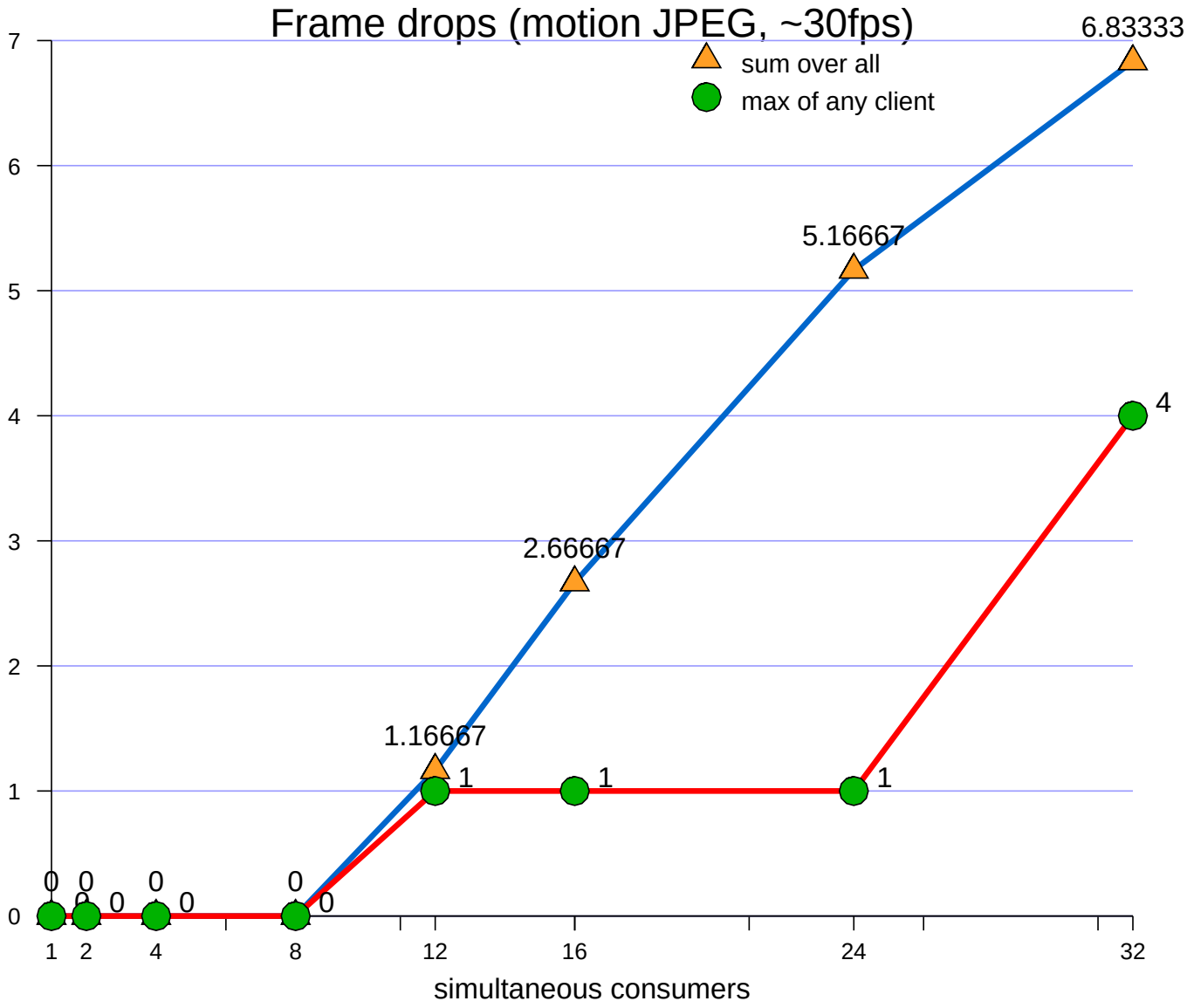
Increasing items in the channel

Local Channel Gets



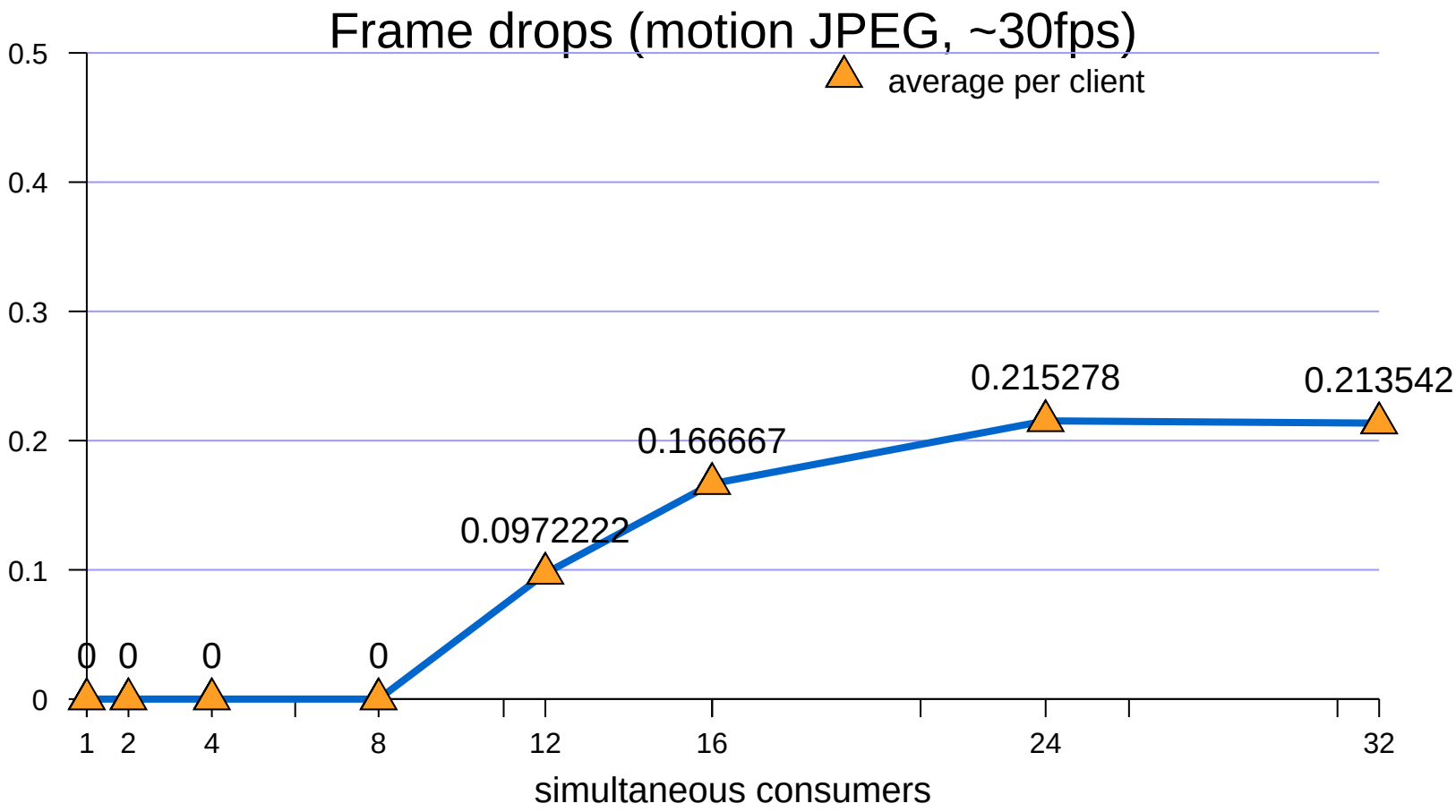
Increasing items in an interval

Remote Channel Gets



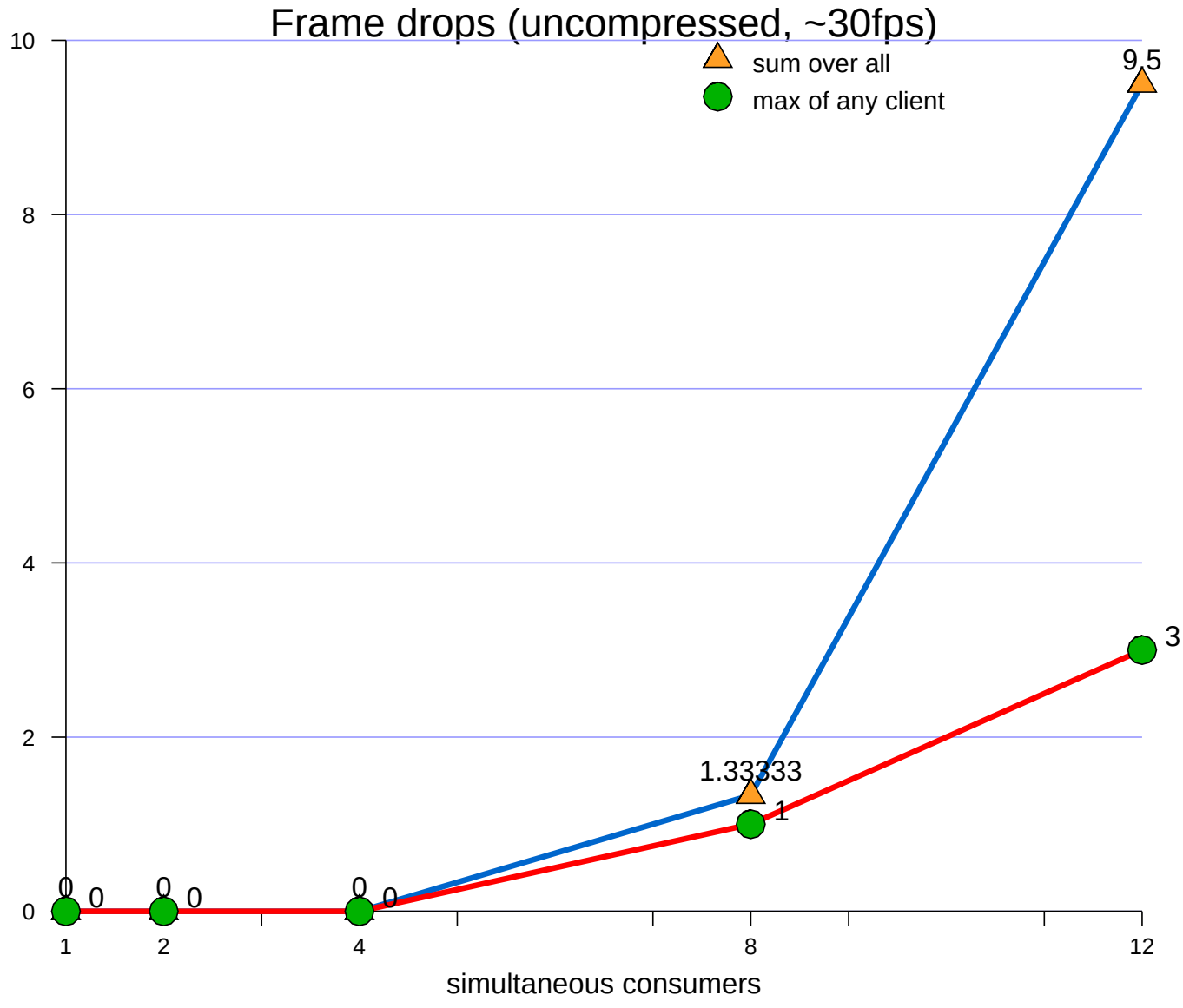
Increasing consumers – MJPEG (sum and max)

Remote Channel Gets



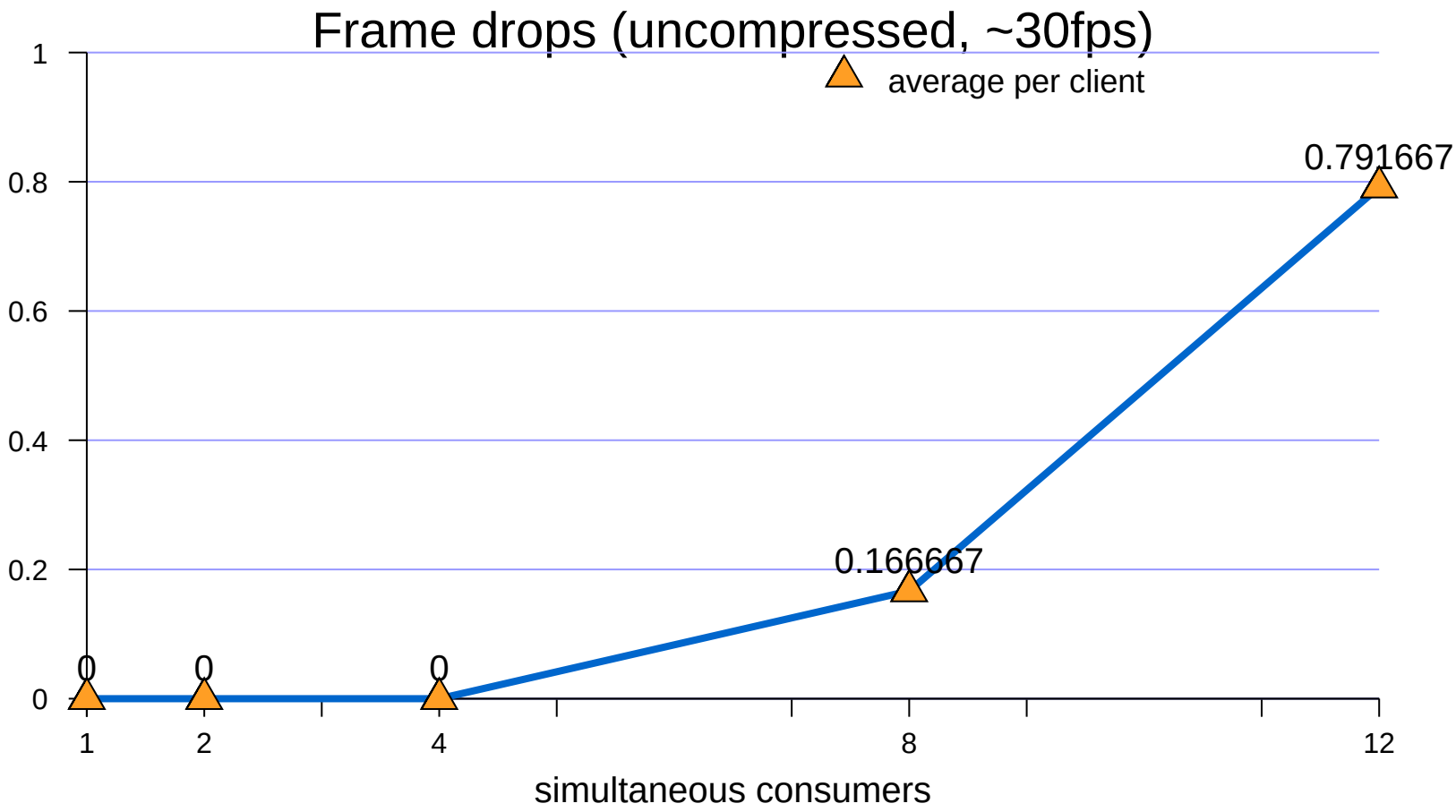
Increasing consumers – MJPEG (average)

Remote Channel Gets



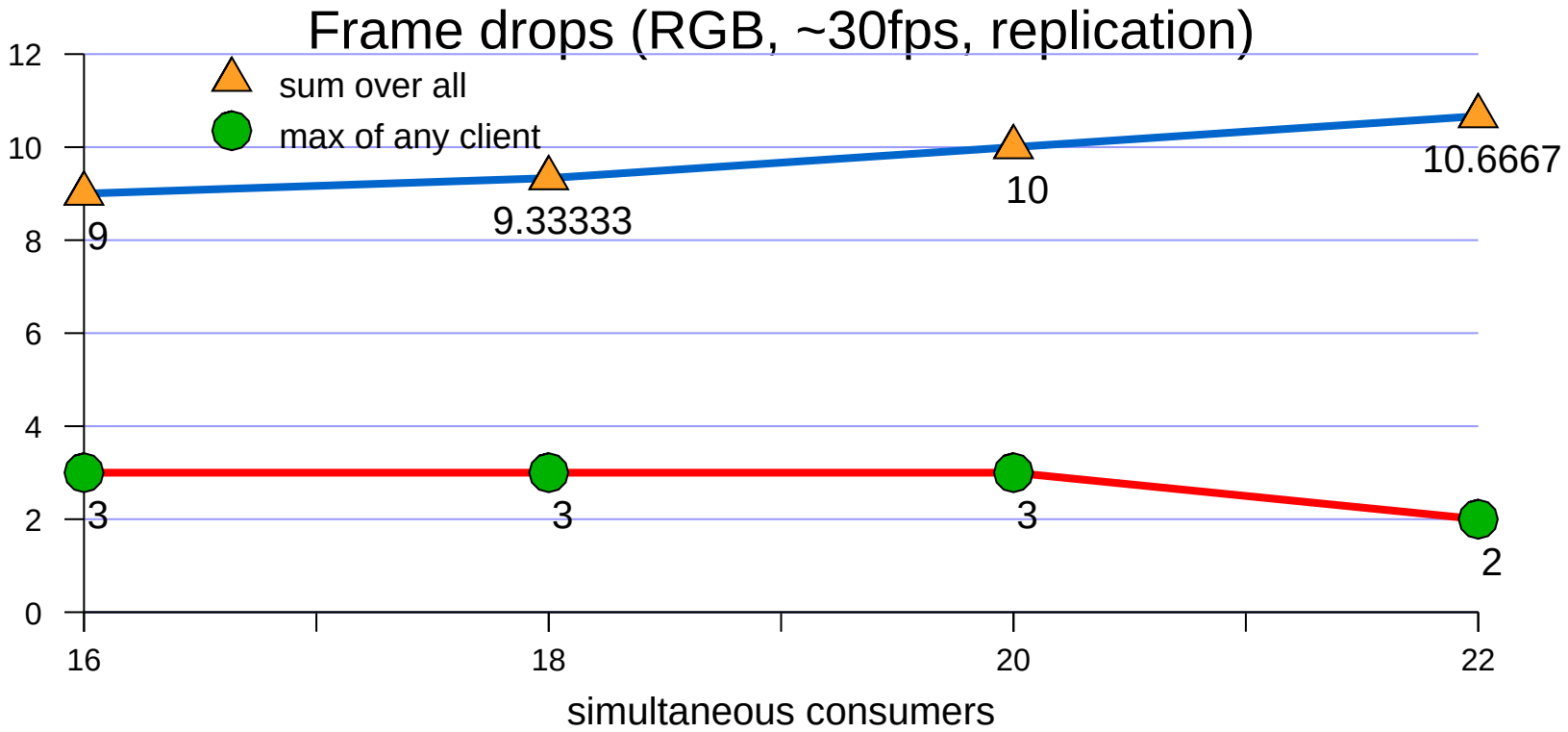
Increasing consumers – RGB (sum and max)

Remote Channel Gets



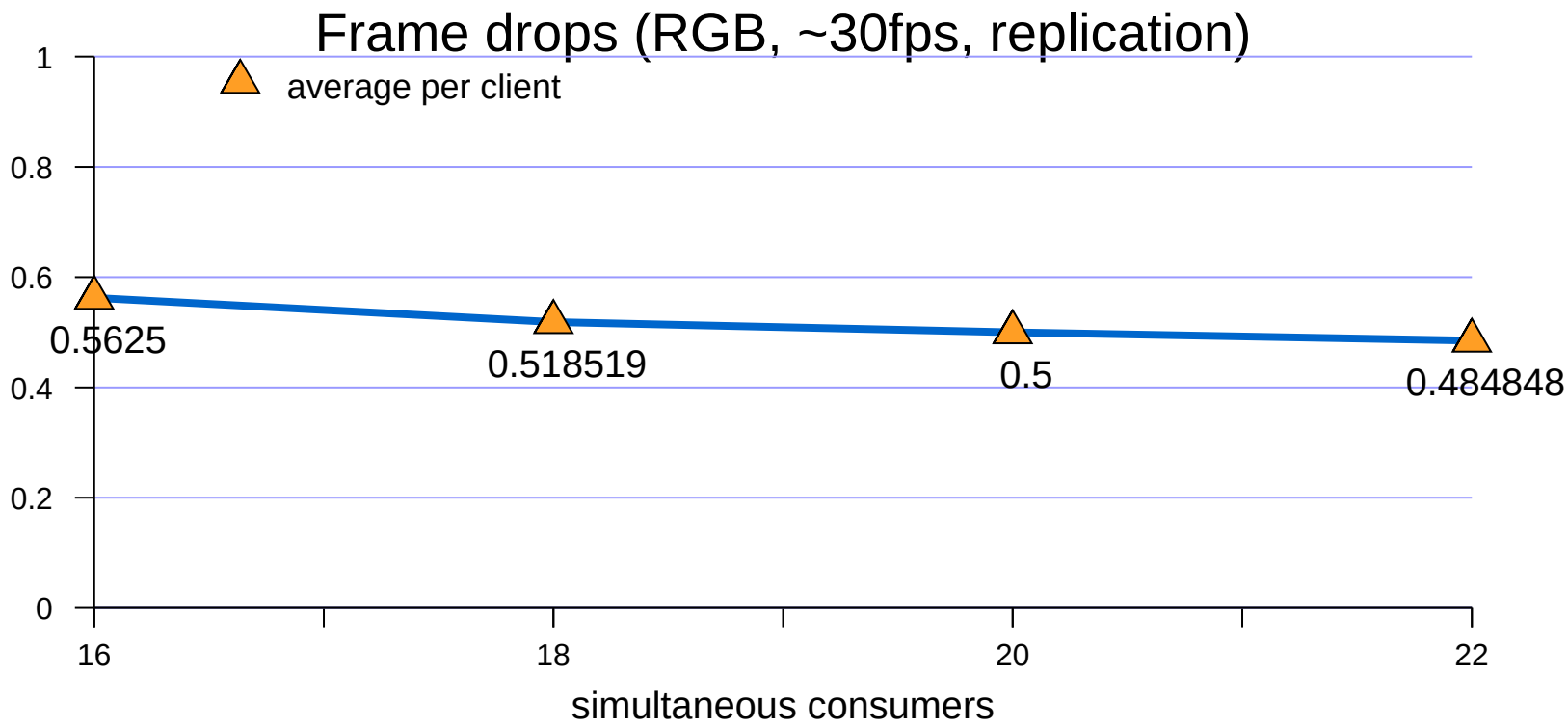
Increasing consumers – RGB (average)

Remote Channel Gets



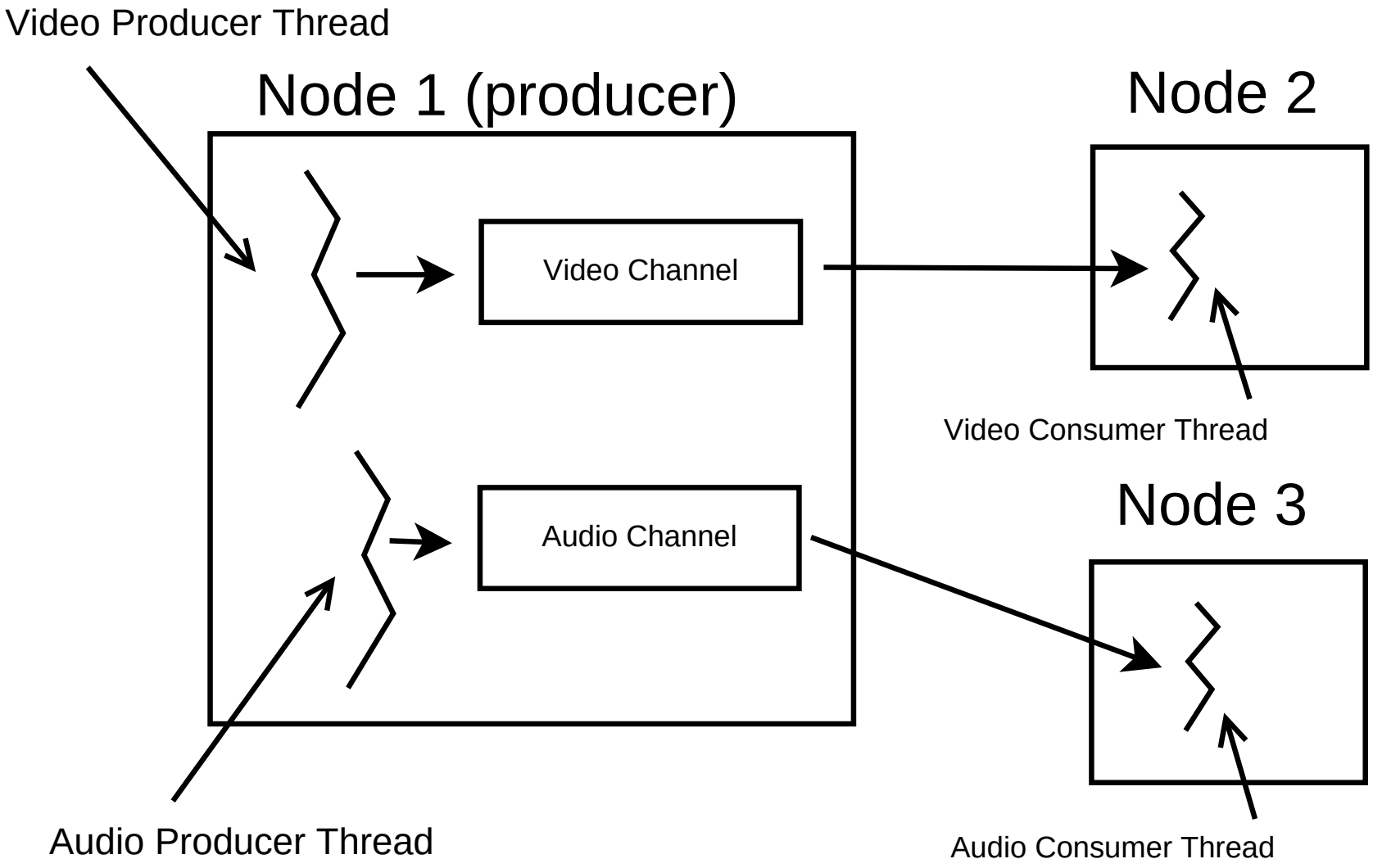
Increasing consumers – Replicated (sum and max)

Remote Channel Gets



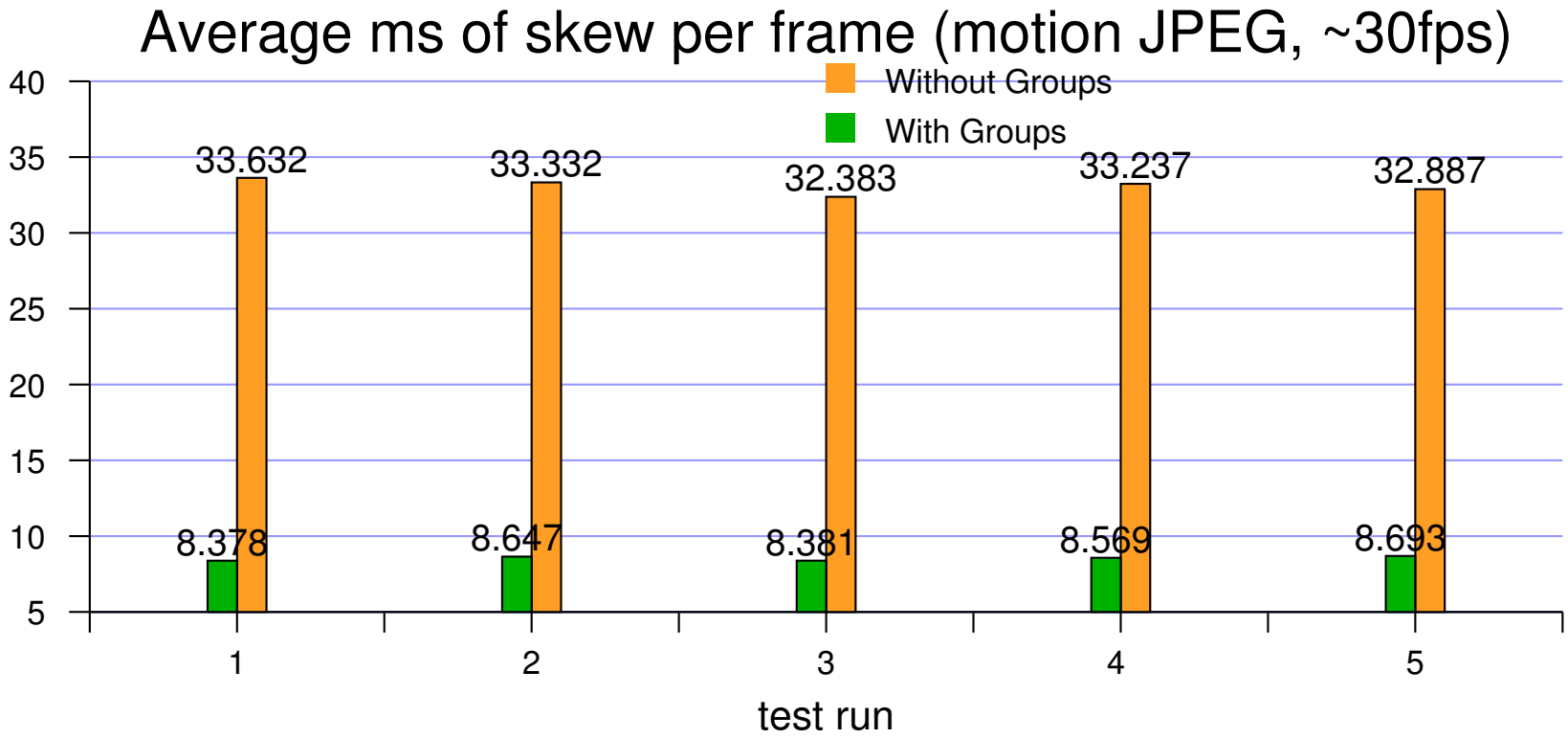
Increasing consumers – Replicated (average)

Channel Group

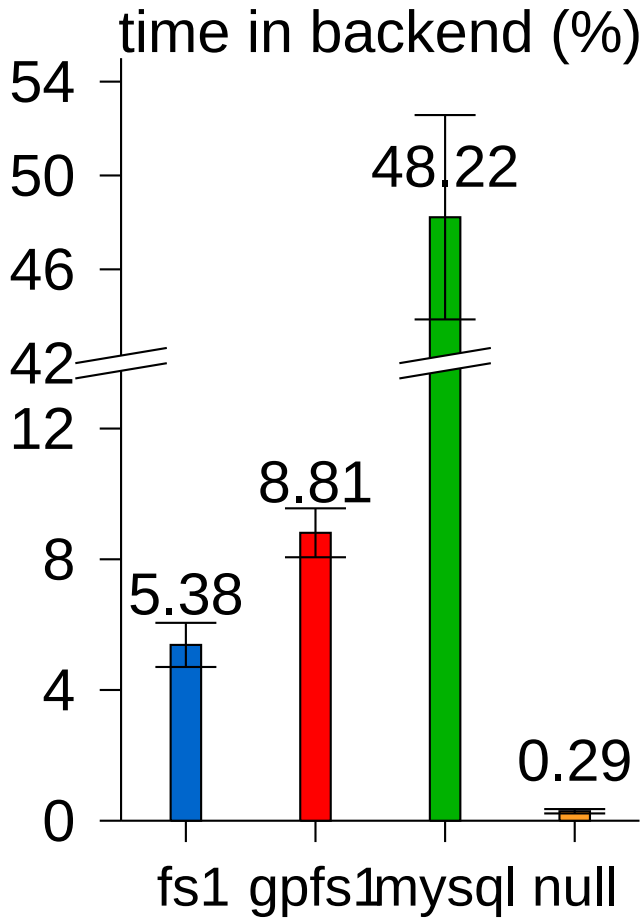


Experimental Setup

Channel Group

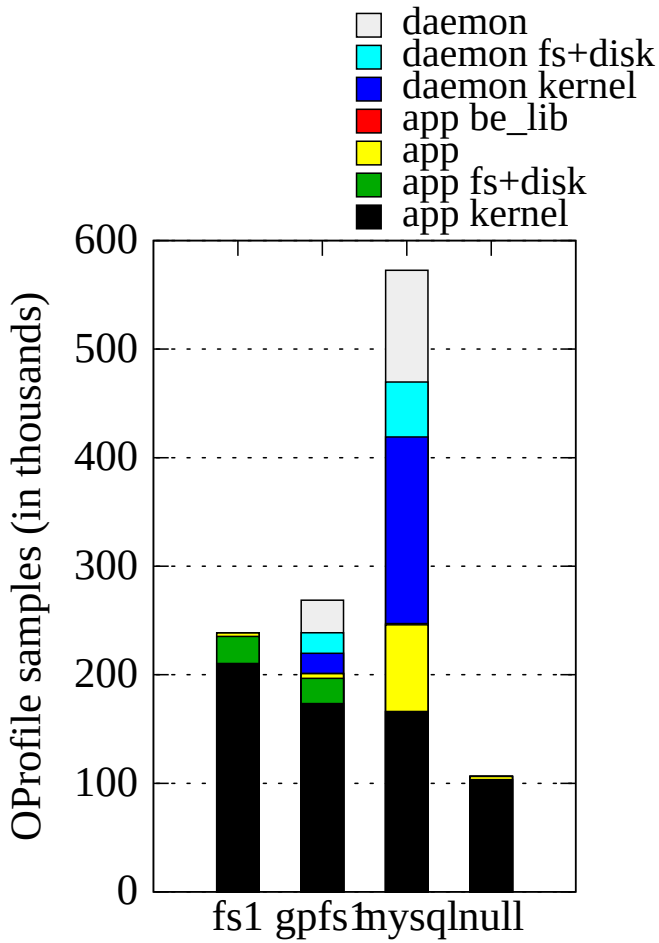


Backend Comparison



Time spent in backend

Backend Comparison



System-wide time

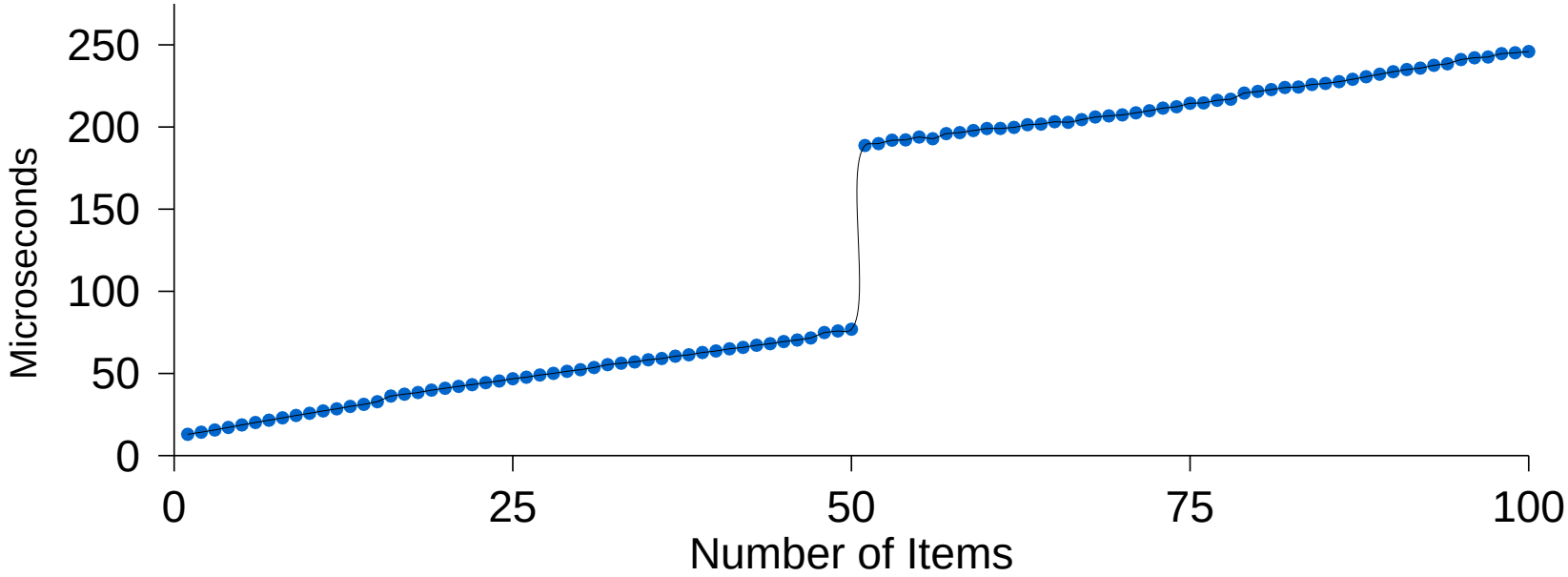
Backend Comparison

Backend	`fs1`	`gpfs1`	`mysql`	`null`
Total Instructions (Millions)	128	143	5,477	126
Instructions in Backend	1.99%	1.83%	97.50%	0.03%

NOTE: *Instructions in Backend* percentages do not include initialization instructions.

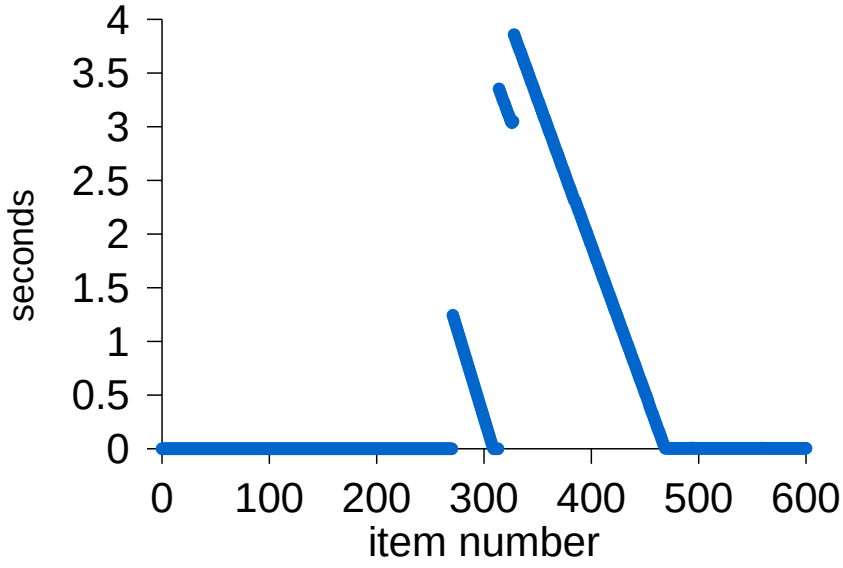
Callgrind (single producer)

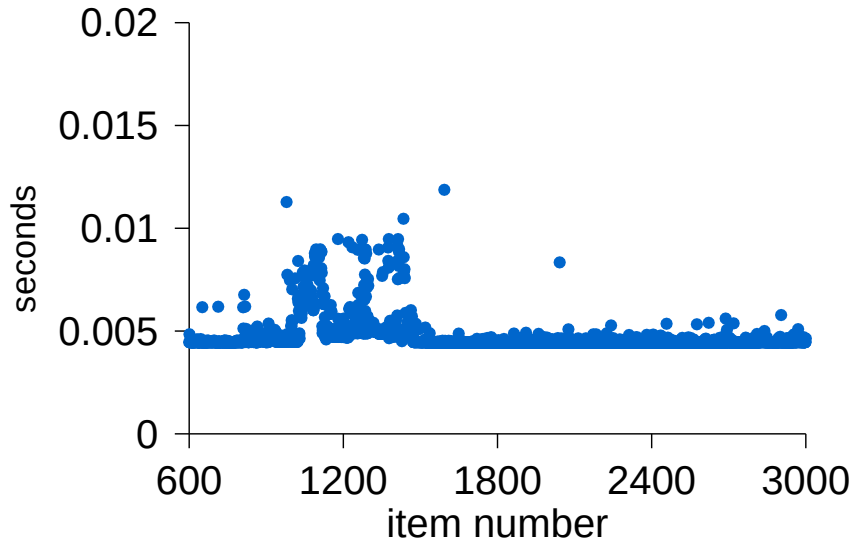
Overhead of Storage Subsystem



$\sim$118 μsecs

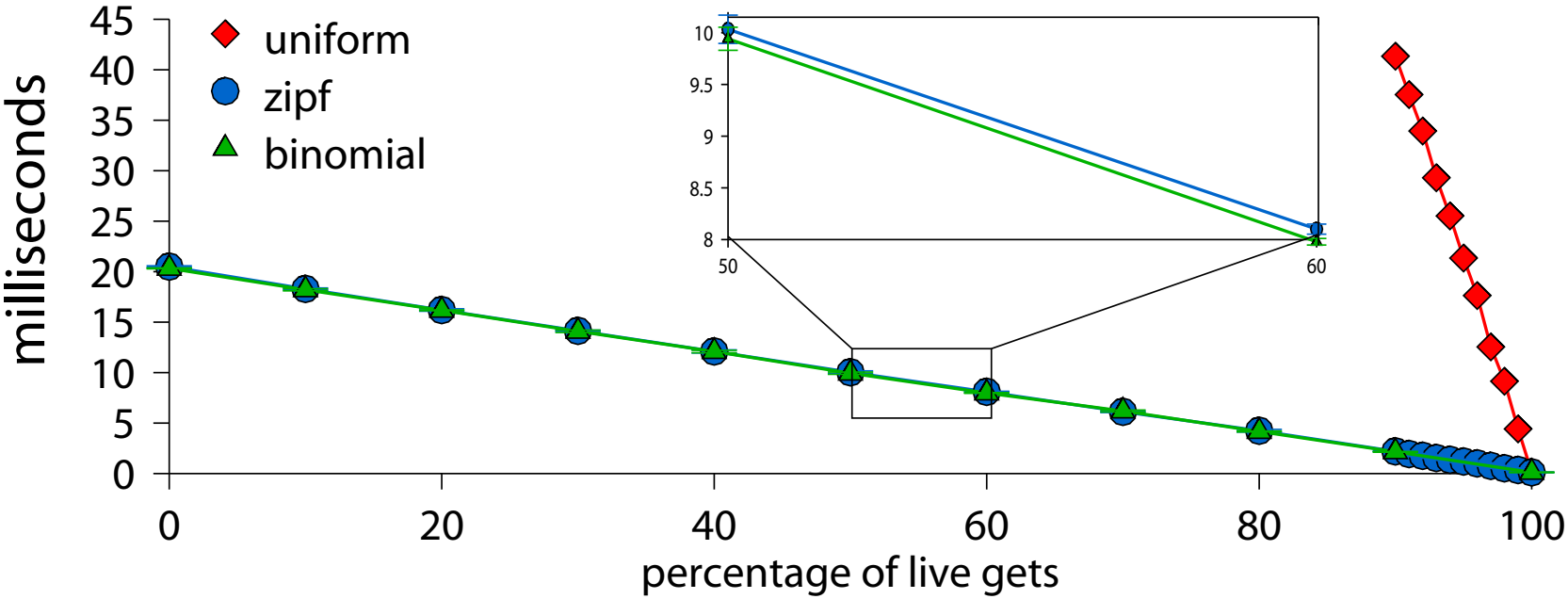
Persistence Automatic Load Adjustment





- Per-item storage latency
- Automatically adjusting to overload
- Dynamically vary pickling handler

Mixed Workload by Distribution



Per-get time with historical query distribution

Unary Feature Detector – main

```
int main(int argc, char **argv) {
  rt_sys_handle_t *rtsh;
  conn_endpt_t inchan, outchan;
  cmdargs_t args;

  // Parse command-line args
  parse_args(&args);

  // Connect to front end and contact super-nodes.
  rtsh = INIT_RT(argv[1]);

  // Get channel descriptor.
  inchan = ADD_CHAN(rtsh, args.input_name);
  outchan = ADD_CHAN(rtsh, args.output_name);

  // Run the transform function
  transform(rtsh, &args, &inchan, &outchan);
  // ...
```

Unary Feature Detector – Transformation

```
void transform(rt_sys_handle_t *rtsh, args_t *args,
               cendpt_t in_ch, cendpt_t out_ch) {
  // Create local copy of in_ch to in
  // ...

  while(!done) {
    // Get one item
    item->status = CONN_GET_1_I(in, &low, &up, item);

    // Successful get
    if(item->status == 0 && item->buf_len >= 1) {
      // Process item, result in buf2

      rval = CONN_PUT(out_ch, buf2,
                      buf2_sz, &item->ts);
    // ...
```

Unary Feature Detector – Optical Flow Example

```
decode_vid_hdr(item->buf, &height, &width, &bpp, &c);

// make grayscale
rgb_to_grayscale(buf3, DATA_OFFSET(item->buf),
                 width, height, bpp);

// halve framerate
if(i & 1 == 1) {
  goto skip;
}

// do OpenCV optical flow
n = detect_optflow1(buf3, width, height, buf2);

// ... send results in buf2 ...
```

Historical Query Generator

```
while(true) {
  // Generate a random time interval based
  // on rp's params
  random_gen(&lower, &upper, percentage, &rp);

  // Get rp.n items
  rval = conn_get_n(chan, &lower, &upper,
                    bufs, bufsizes, rp.n, metadata);

  ...
}
```

Binary Feature Detector – main

```
// ...
// Parse command-line args
// Connect to front end and contact super-nodes.

// Get channel descriptor.
inchan = ADD_CHAN(rtsh, args.input_name);
inchan2 = ADD_CHAN(rtsh, args.input_name2);
outchan = ADD_CHAN(rtsh, args.output_name);

// Create channel group
int ch_nums[2] = {inchan.chan_num, inchan2.chan_num};
MAKE_GROUP(rtsh, ch_nums, gr_nums, 2, 1);

// ... Create background get thread ...
// Run the transform function
```

Binary Feature Detector – Background Fetching Thread

```
// ...
while(!done) {
  // Get one item
  item->status = CONN_GET_1_I(in, &low, &up,
                              item, gr_nums[1]);

  // Successful get
  if(item->status == 0 && item->buf_len >= 1) {
    low = TIME_ADD(&item->ts, &one_micro);

    ATOMIC_PUT(&globbg, item->buf);

    // ...
  }
}
```

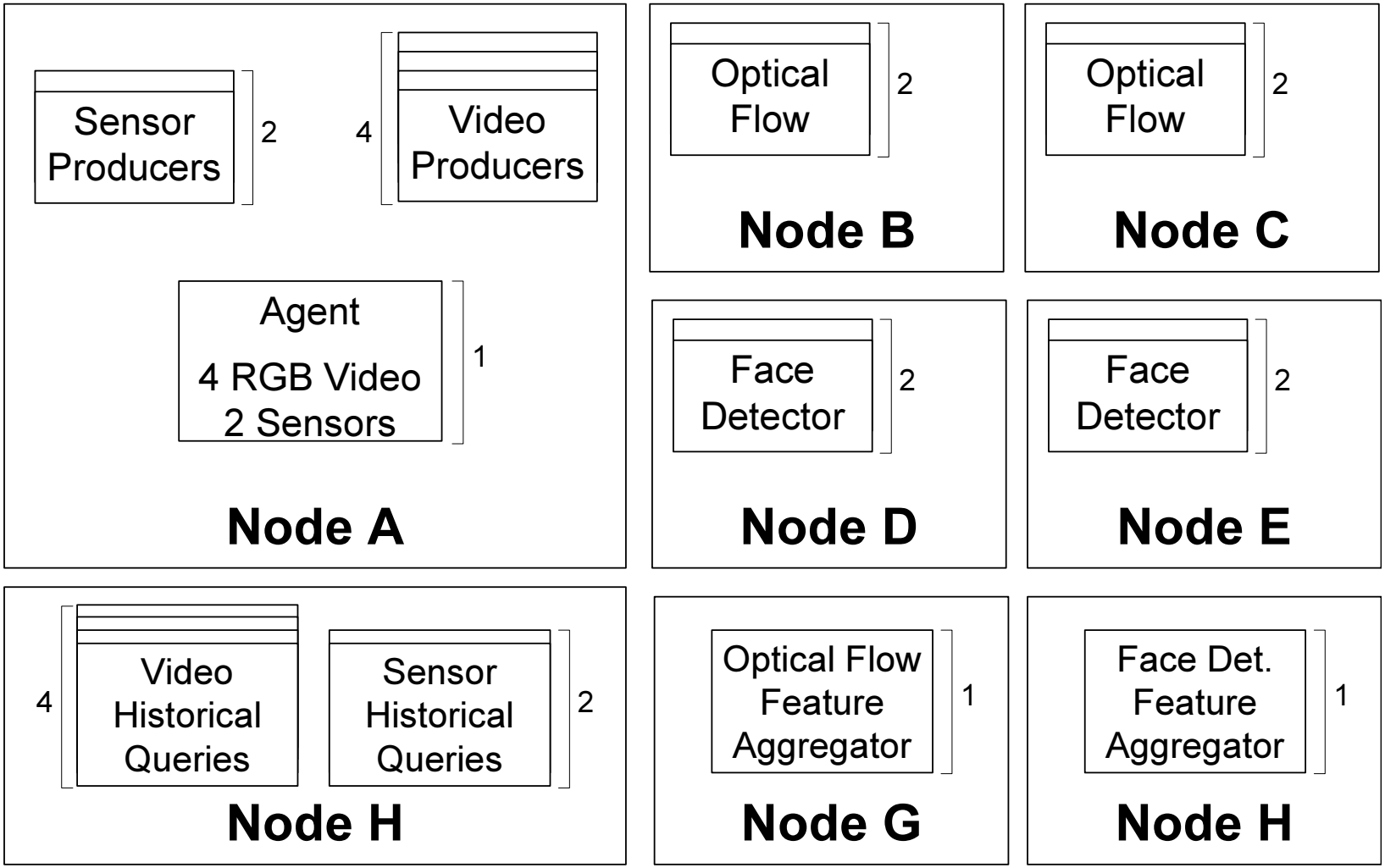
Binary Feature Detector – Foreground Separation

```
decode_vid_hdr(item->buf, &height, &width, &bpp, &c);

// grab current background
ATOMIC_GRAB(&buf, &globbg);
cvSetImageData(&bgImg, DATA_OFFSET(buf), width*3);

// do foreground separation
res = detect_fgdata1(DATA_OFFSET(item->buf), &bgImg,
                     &tmpIn, width, height, buf2);

// ... send results in buf2 ...
```

Binary Feature Detector – Camera Change Example

```
decode_vid_hdr(item->buf, &height, &width, &bpp, &c);

// histogram scene change
res = detect_schist1(DATA_OFFSET(item->buf),
                     &lastHist, width, height,
                     corr_hist, ncorr_hist,
                     &curr_start, 0.5);
...

if(res > threshold) {
  upper = NOW();

  // fetch BG images from bg_chan to confirm
  lower = TIME_SUB(item->ts, TIME10MS);
  CONN_GET_N(bg_chan, &lower, &upper,
             bufs, bufsizes, 10,
             NULL, gr_nums[1]);
  // ...
```

Airport Surveillance



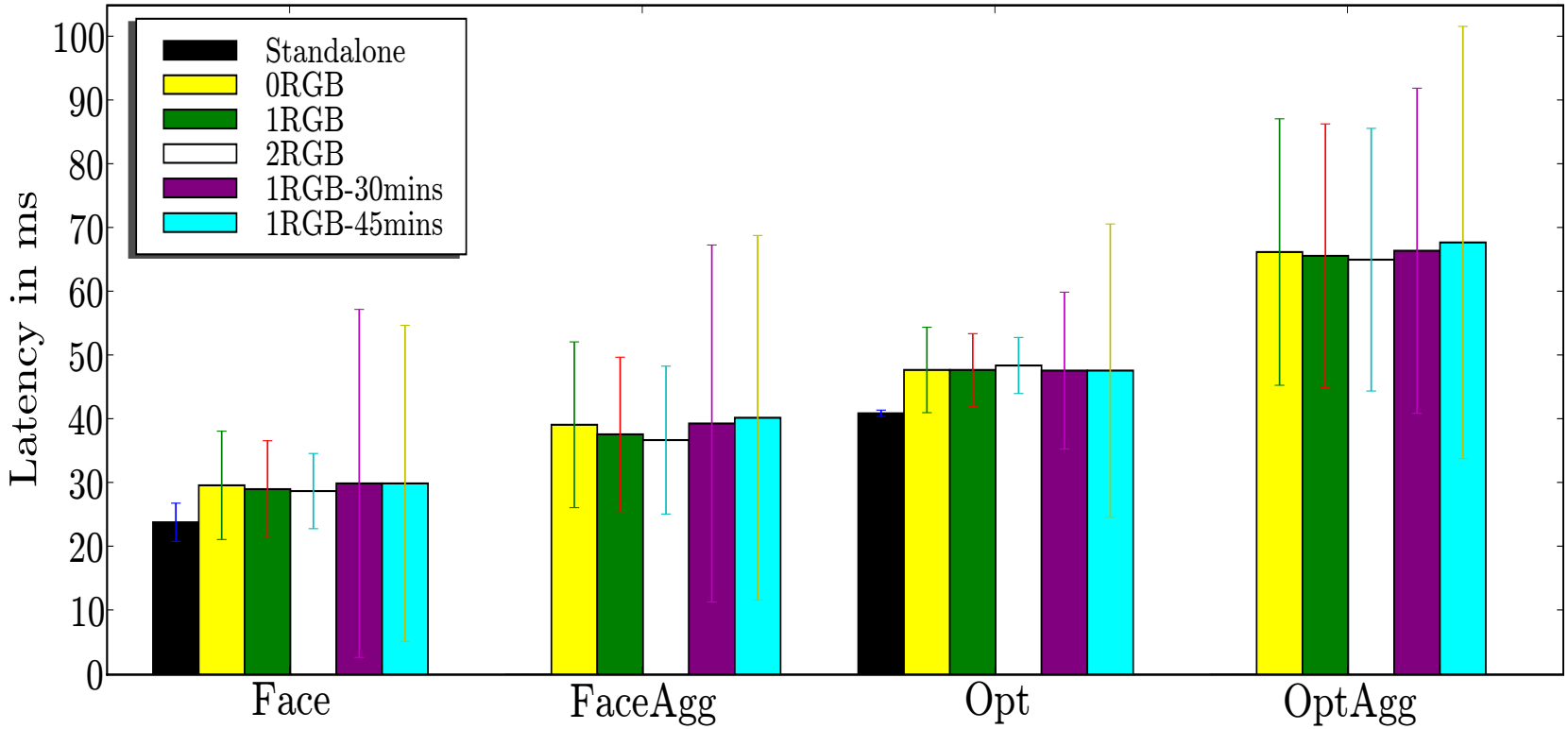
Topology

Airport Surveillance

Component	Configuration	Total	+Nodes
Agent	1 per node, hosts 4 vid. / 2 sensor	2	2
Producers	6 per ASAP node, one per stream	12	-
Historical Query	6 per dedicated node	12	2
Face Detection	2 per dedicated node	8	4
Optical Flow	2 per dedicated node	8	4
Face Aggregator	1 per dedicated node	1	1
Optical Flow Aggregator	1 per dedicated node	1	1

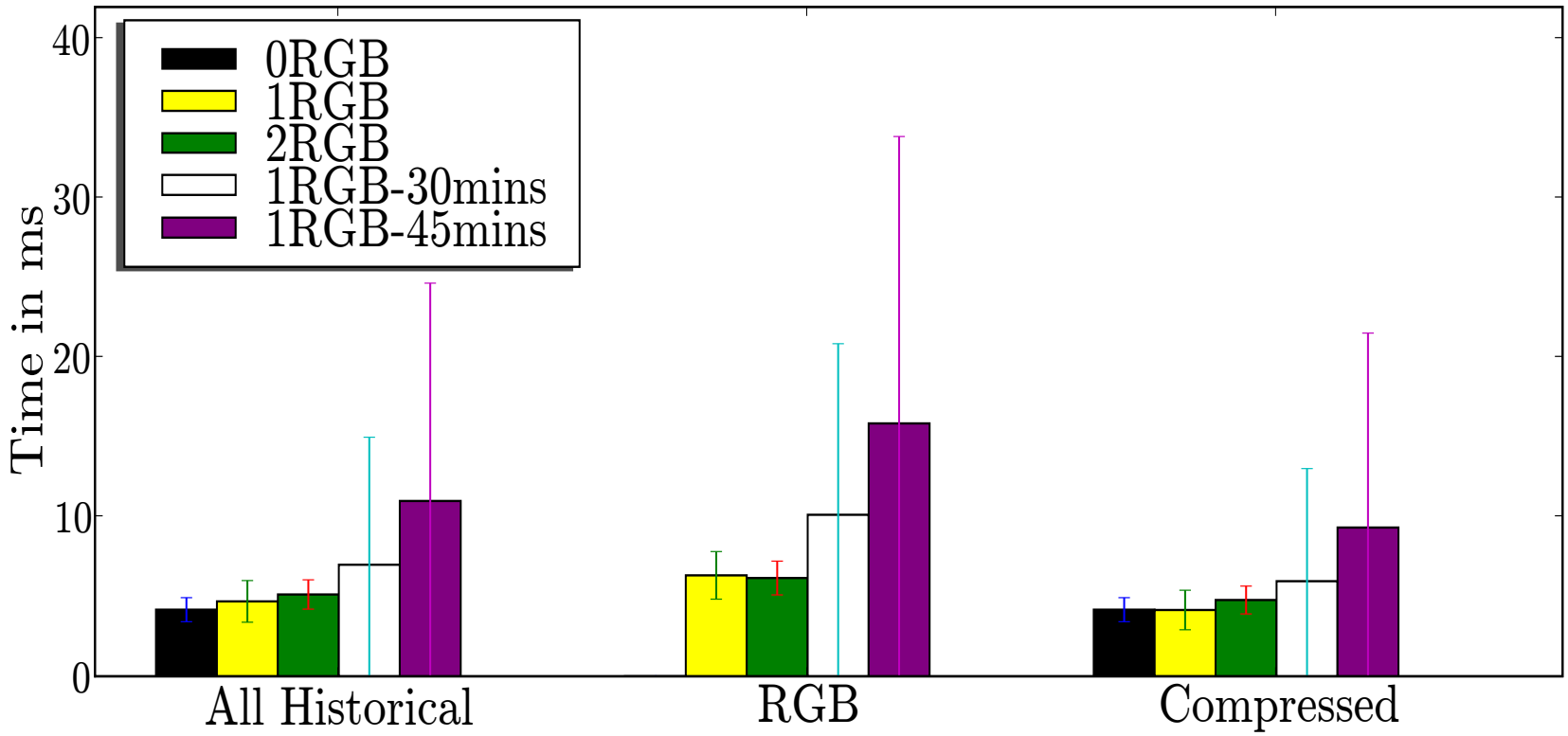
The "+Nodes" column specifies the additional nodes required by each column.

Airport Surveillance



Component Latency in ms.

Airport Surveillance



Query Time in ms.

Airport Surveillance

Data	Video	Face	Optical Flow	Sensor
Payload	230,408	128	128	1024
Response Header	64	64	64	64
Item Metadata	20	20	20	20
Total System Data	84	84	84	84
Total	230,492	212	212	1108
Overhead %	0.036%	39.62%	39.62%	7.58%

System Data Overheads in bytes

Airport Surveillance

Data	Video	Face	Optical Flow	Sensor
Payload	230,408	128	128	1024
Response Header Len	2	2	2	2
Response Header	13	13	13	13
Data Header Len	2	2	2	2
Data Header	4	3	3	3
Item Metadata Len	2	2	2	2
Item Metadata[1]	15-21	14-20	14-20	14-20
Total System Data	38-42	36-40	36-40	36-40
Max Total	230,450	168	168	1064
Overhead %	0.018%	23.81%	23.81%	3.76%

System Data Overheads (Variable Length)

Port Asset Tracking – Drools Syntax Example

```
RULE "Syntax Example"
  WHEN
       [condition 1]
       [condition 2]
       ...
  THEN
       [action 1]
       [action 2]
       ...
END
```

Port Asset Tracking – Truck Entering Port

```
RULE "Truck Entering Port"
  WHEN
    $tee : TruckEnterEvent()
  THEN
    log($tee)
END
```

Port Asset Tracking – Truck Leaving Port

```
RULE "Truck Leaving Port"
  WHEN
    $tle : TruckLeaveEvent($tsl: tmstamp, $id1: id)
    $tee : TruckEnterEvent($tse: tmstamp, id == $id1)
  THEN
    diff = ($tsl.getTime() - $tse.getTime()) / 1000)
    log("Truck %d leaving at %d secs", $id1, diff)
    retract($tle)
    retract($tee)
    insert(new TruckCycle($tee, $tle, diff))
END
```

Port Asset Tracking – Phantom Truck Leaving Port

```
RULE "Phantom Truck Leaving Port"
  WHEN
    $tle : TruckLeaveEvent($id1: id)
    not ( exists (TruckEnterEvent(id == $id1)) )
  THEN
    log("No enter event: phantom truck %d", $id1)
    // diagnose anomaly
    diagnosePhantom($tle)
END
```

Port Asset Tracking – Phantom Truck Actions

```
// rule "Phantom Truck Leaving Port" actions
void diagnosePhantom(TruckLeaveEvent tle) {
  Long ts = tle.getTimestamp().getTime() / 1000;

  // based on estimated truck timing, find potential
  // video of entry
  Long tentry = ts - getAvgCycleTime();
  Long tdelta = getAvgCycleTime() / 10;
  boolean done = found = false;

  while(!done && !found) {
    // ± 10% around potential entry
    Item[] i = entryCam.getItems(tentry - tdelta,
                                 tentry + tdelta);

    // show human operator
    OpOutput oo = showOperator(i);

    // using operator feedback, continue searching
    tentry = oo.getTEntry();
    tdelta = oo.getTDelta();
    found = oo.isFound(), done = oo.isDone();
  }

  // ...
```

Port Asset Tracking – Truck Leaving Port Late

```
RULE "Truck Leaving Port Late"
  WHEN
    // >55 mins is late (time is in seconds)
    $tc: TruckCycle(time > (60*55))
  THEN
    log("Truck Left Late!")
    // diagnose anomaly
    diagnoseLateDeparture($tc)
END
```

Port Asset Tracking – Late Departure Actions

```
// rule "Truck Leaving Port Late" actions
void diagnoseLateDeparture(TruckCycle tc) {
  Long len = tc.getTruckLeaveEventTime() -
             tc.getTruckEnterEventTime();
  Long tdelta = len / 10;

  // do a binary search for the truck delay; find
  // the first point where the truck starts running late

  // start at the middle of the journey
  Long ts = tc.getTruckEnterEventTime() + (len / 2);

  while(!done) {
    // get appropriate channel for time in delivery cycle
    chan = getVideoChannelByTime(tc, ts);

    // ...
    Item[] i = chan.getItems(ts - tdelta, ts + tdelta);

    // show operator video
    OpOutput oo = showOperator(i);

    // modify search based on feedback
    // ...
```

Port Asset Tracking – Truck Missed Checkpoint

```
RULE "Truck Missed Checkpoint"
  WHEN
    // if we have a gap in checkpoints
    $ce1 : CheckptEvent($id1: id, $point: point > 1)
    not( exists ( CheckptEvent(id == $id1 &&
                               point == ($point-1)) ))
    // collect all prior checkpoints into a list
    $prior : ArrayList() from
             collect( CheckptEvent(id == $id1,
                                   $p2: point < $point) )
    // find the most recent checkpoint before missing
    $lp : Number() from
          accumulate( CheckptEvent(id == $id1,
                                   $p2: point < $point),
                      max($p2) )
  THEN
   log("Truck %d missing checkpoint #%d", $id1, $point-1)
   log("Last checkpoint #%d: %s", $lp, $prior)
   // diagnose anomaly
   diagnoseMissedCheckpoint($ce1, $prior, $lp)
END
```

Port Asset Tracking – Missed Checkpoint Actions

```
// rule "Truck Missed Checkpoint" actions
void diagnoseMissedCheckpoint(CheckptEvent ce,
                              ArrayList<CheckptEvent> lst,
                              Long lastPointId) {
  HashMap<Long, CheckptEvent> map = pointListToMap(lst);
  CheckptEvent last = map.get(lastPointId);

  // ...

  Channel c = getVideoChanByCheckpt(lastPointId+1);

  // get data between two good checkpoints
  Item[] i = c.getItems(last.getTimestamp(),
                        ce.getTimestamp());

  // use video for diagnosis

  // ...

}
```

Code Complexity

Component	SLOC	Generated SLOC
Supernode (Erlang)	510	1006
Front-end (Python)	192	-
Utilities (C)	1544	-
IDL (Sun RPC $\rightarrow$ C)	231	1084
C Client Library (C)	4507	-
5 Storage Backends (C)	2030	-
Storage Backend Common (C)	250	-
Backend IDL (Sun RPC $\rightarrow$ C)	45	282
C Runtime Totals	8449 + 276 (IDL)	1366
Java Client Library (Java)	1663	-

First Generation System

Code Complexity

Component	SLOC	Generated SLOC
`fs1`	771	-
`gpfs1`	568	-
`mysql`	391	-
`pgsql`	271	-
`null`	29	-
Common	250	-
Common IDL	45	282
Total	2280 + 45 (IDL)	282

Storage Backends

Code Complexity

Component	SLOC	Generated SLOC
IDL (Protobuf $\rightarrow$ Java)	115	5692
Java Client Library	1517	-

Second Generation System

Code Complexity

Component	SLOC
Airport Surveillance (C)	1050
Traffic Monitoring (C)	810
Port Asset Tracking (DRL)	75
Port Asset Tracking (Java)	250

Applications

Value of System Philosophy

- *V(A)aaS* – video-analytics-as-a-service
- Live stream analysis in the cloud
- Provides flexibility
- Characteristics:
  - Looser coupling
  - "Black box" components
  - Spans administrative domains
  - Glue between distributed components

Themes

- Transparent vs. Opaque Computation
- Structured vs. Unstructured Streams
- Granularity of Stream Data
- Granularity of Dataflow
- Streams & Computation