Towards General-Purpose Resource Management in Shared Cloud Services

Jonathan Mace, *Brown University*

Peter Bodik, *MSR Redmond*

Rodrigo Fonseca, *Brown University*

Madanlal Musuvathi, *MSR Redmond*

Shared-tenant cloud services

Processes service requests from multiple clients

- ✓ Great for cost and efficiency
- ✗ Performance is a challenge

Aggressive tenants and system maintenance tasks

↓

Resource starvation and bottlenecks

↓

Degraded performance, Violated SLOs, system outages

Shared-tenant cloud services

Ideally

manage resources to provide end-to-end guarantees and isolation

Challenge

OS/hypervisor mechanisms insufficient

- ✗ Shared threads & processes
- ✗ Application-level resource bottlenecks (locks, queues)
- ✗ Resources across multiple processes and machines

Today

lack of guarantees, isolation

some ad-hoc solutions

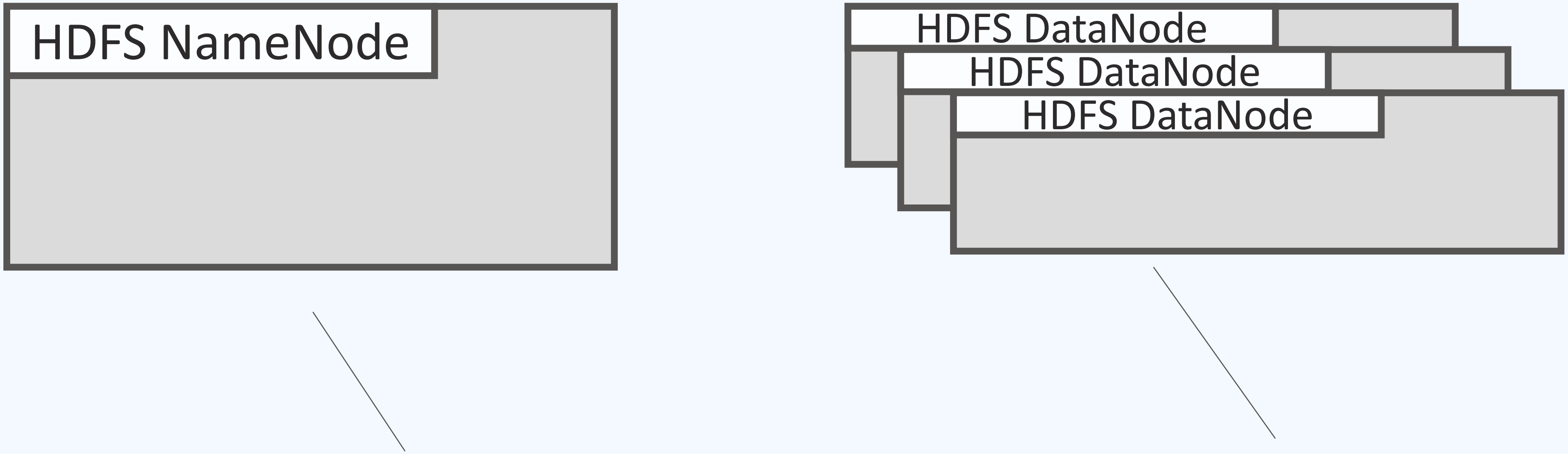
This paper

- 5 design principles for resource policies in shared-tenant systems

- *Retro* – prototype for **principled** resource management

- Preliminary demonstration of Retro in HDFS

Hadoop Distributed File System (HDFS)



HDFS NameNode

HDFS DataNode

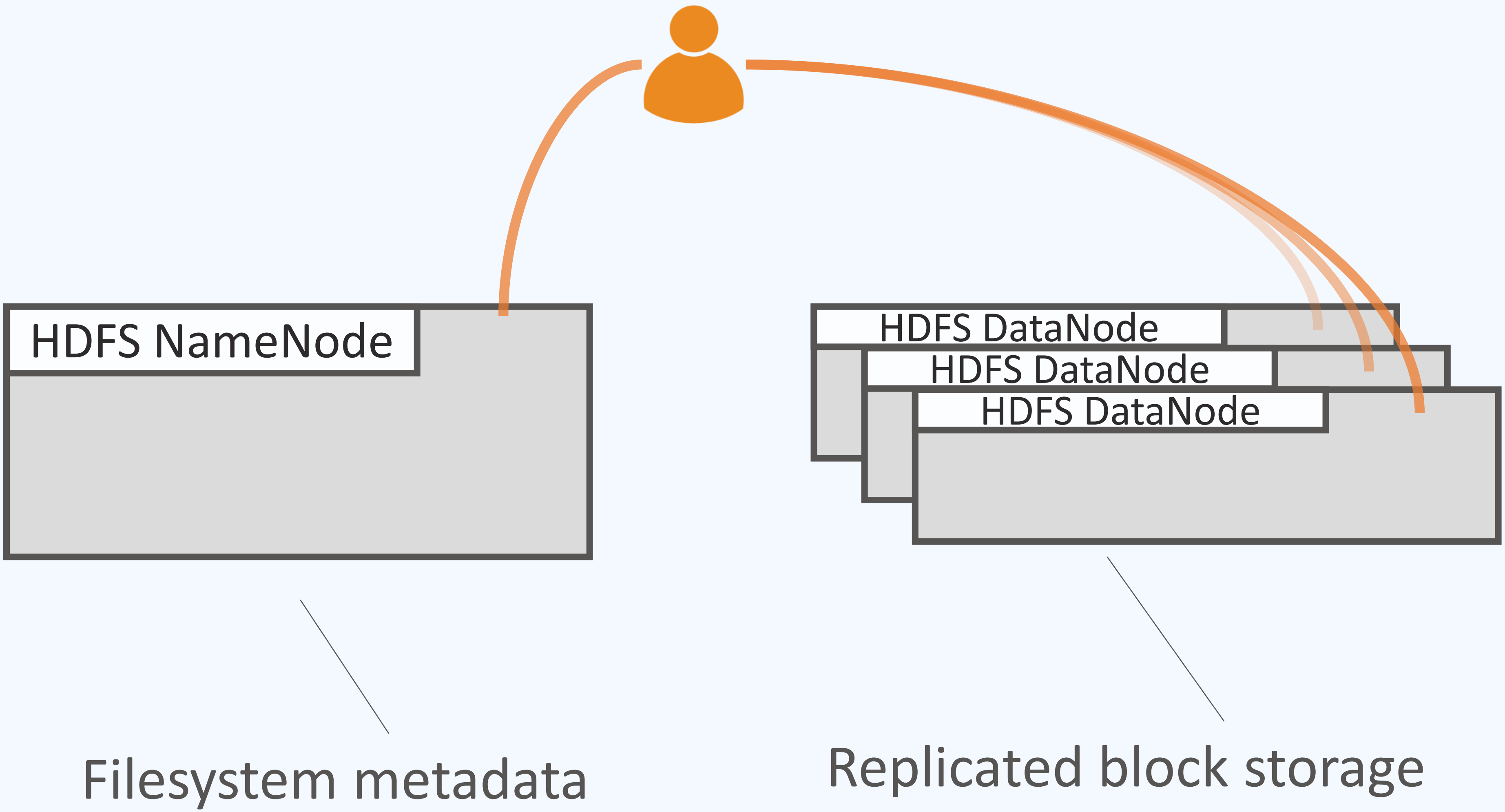
HDFS DataNode

HDFS DataNode

Filesystem metadata

Replicated block storage

Hadoop Distributed File System (HDFS)



HDFS NameNode

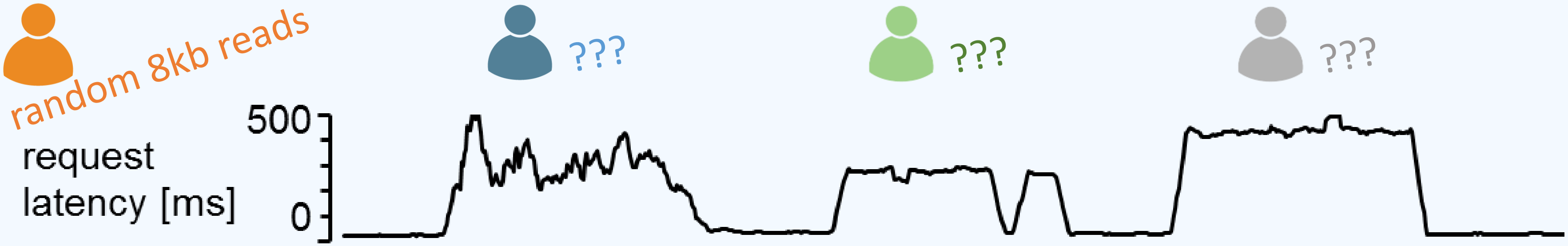
HDFS DataNode

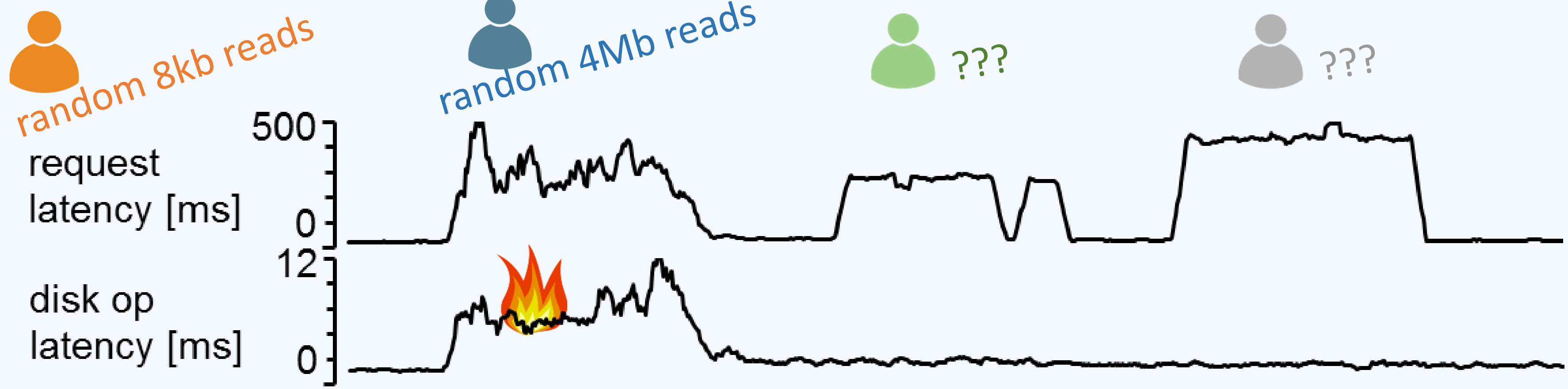
HDFS DataNode

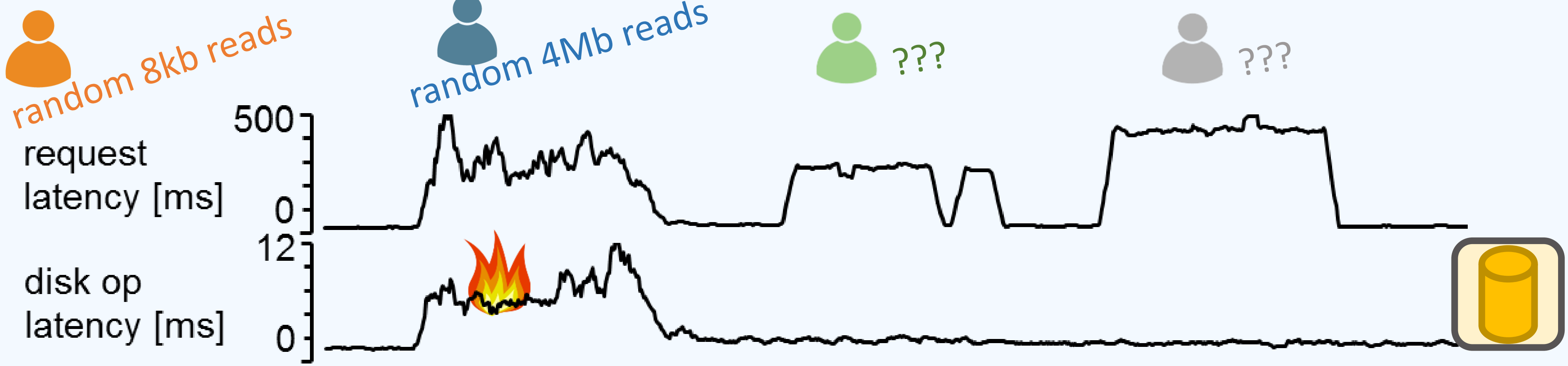
HDFS DataNode

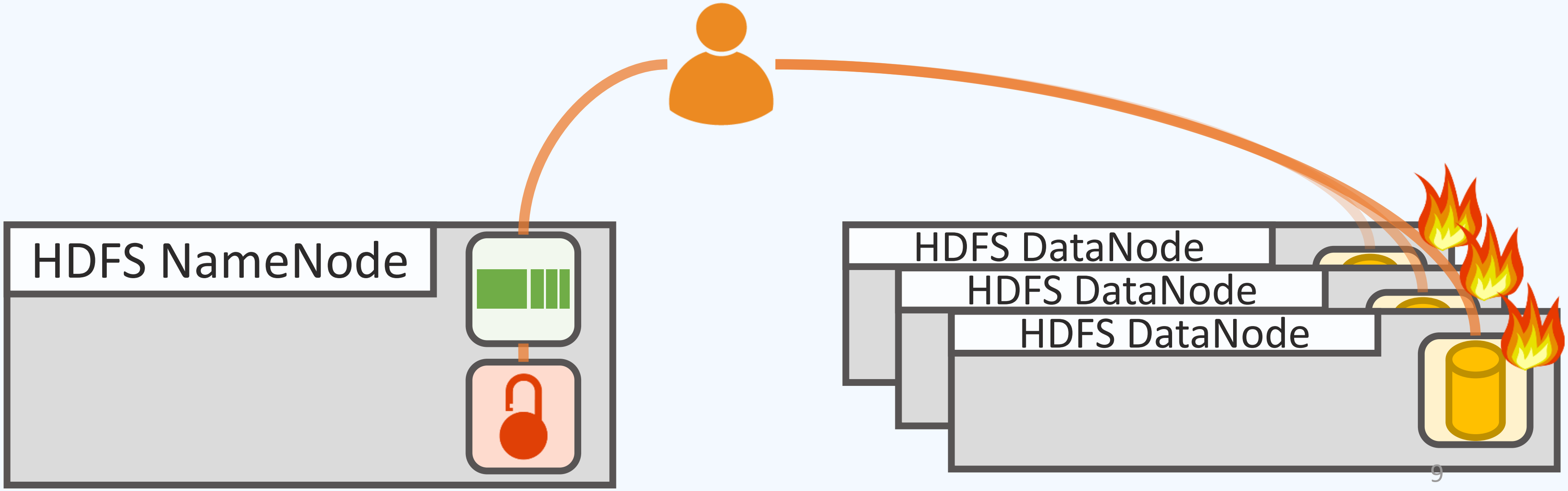
Filesystem metadata

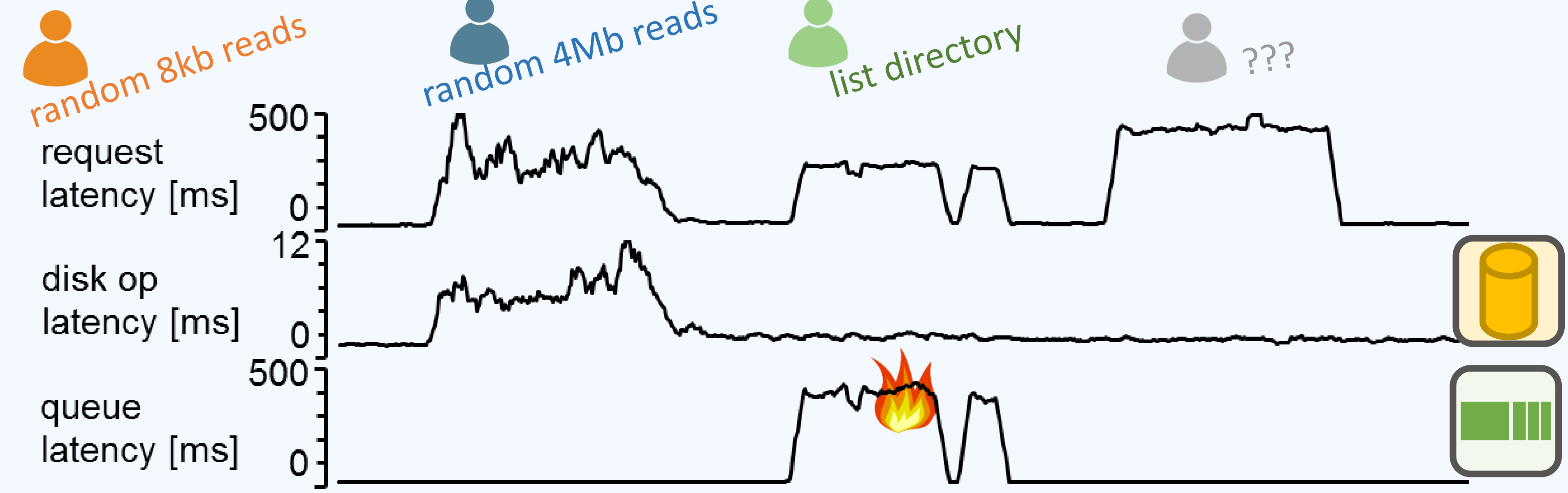
Replicated block storage

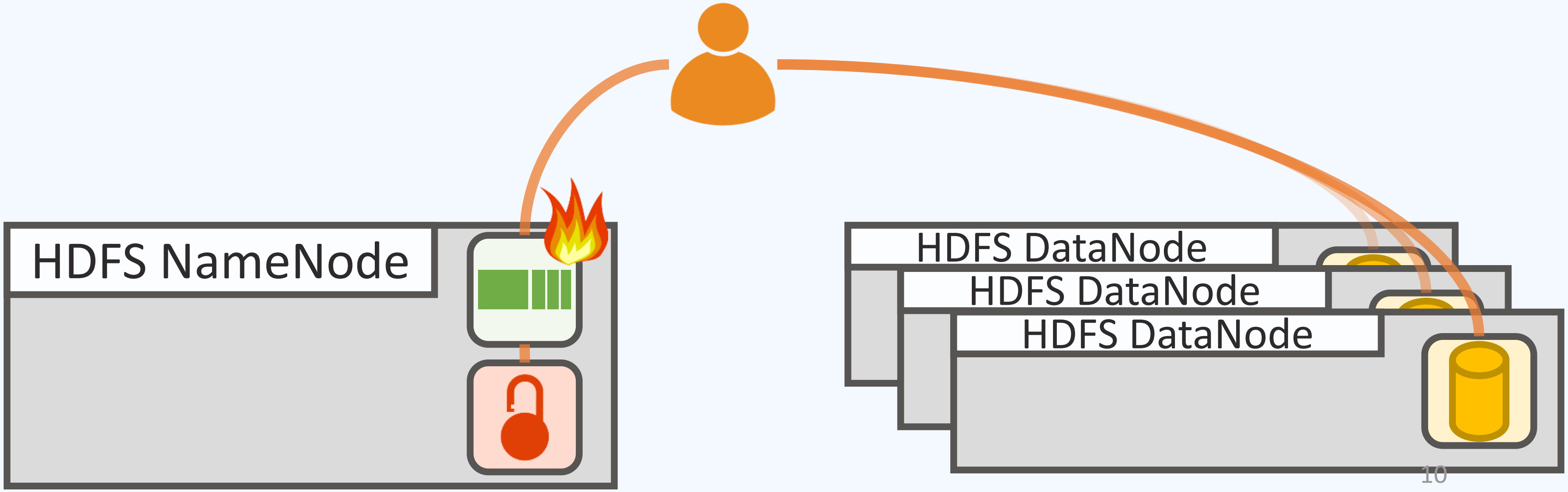

random 8kb reads
???
???
???
request
latency [ms]
500
0

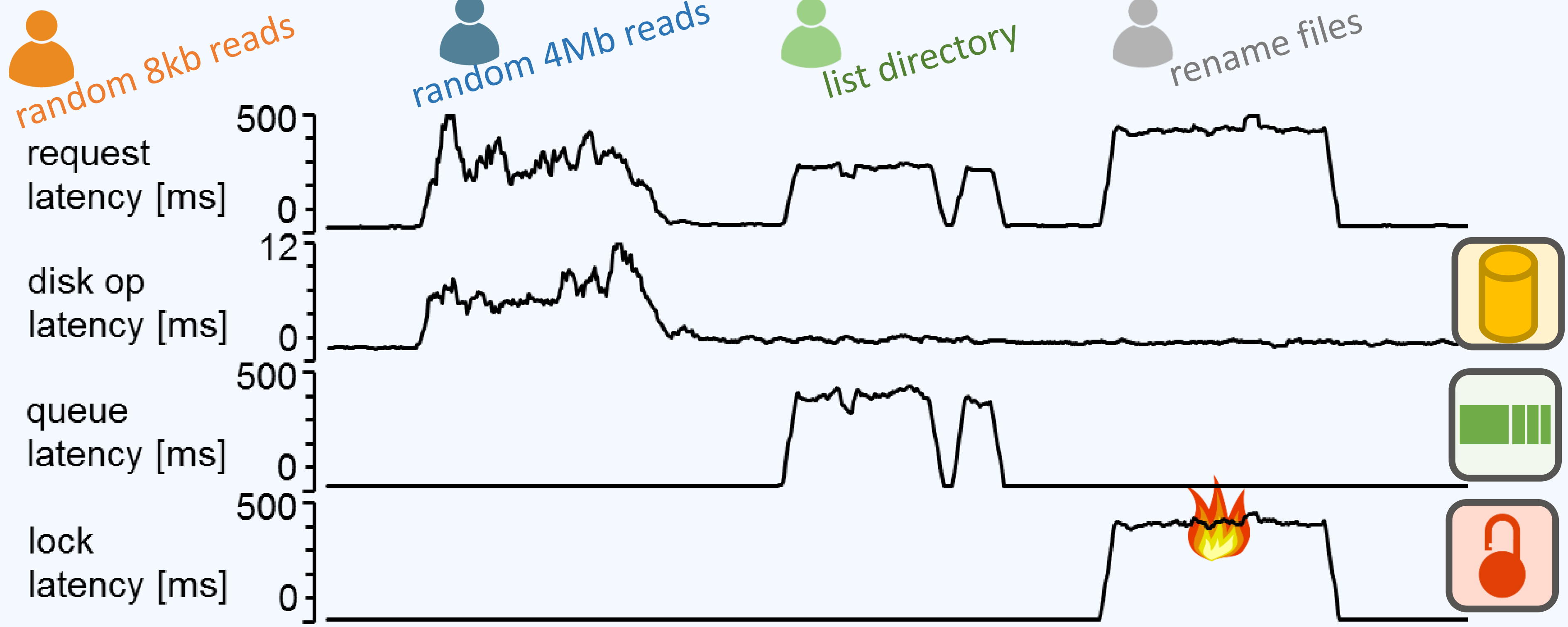

random 8kb reads
random 4Mb reads
???
???
request latency [ms]
500
0
disk op latency [ms]
12
0

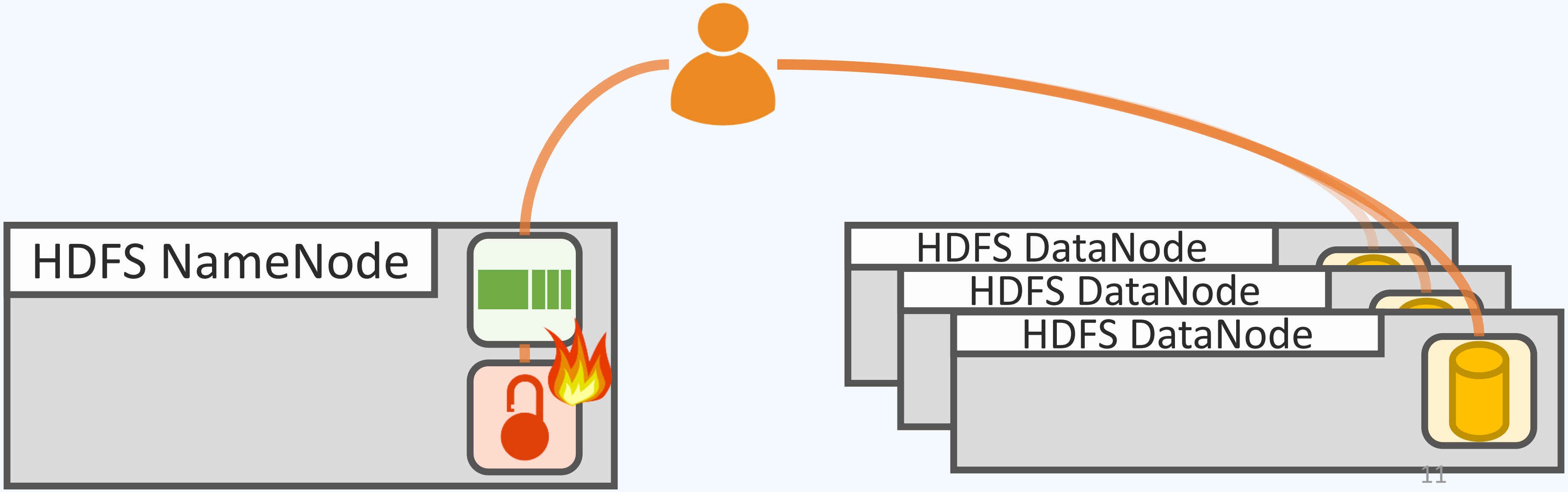

random 8kb reads
random 4Mb reads
???
???
request latency [ms]
500
0
disk op latency [ms]
12
0


HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode


random 8kb reads
random 4Mb reads
list directory
???
request latency [ms]
500
0
disk op latency [ms]
12
0
queue latency [ms]
500
0


HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode




HDFS NameNode

HDFS DataNode

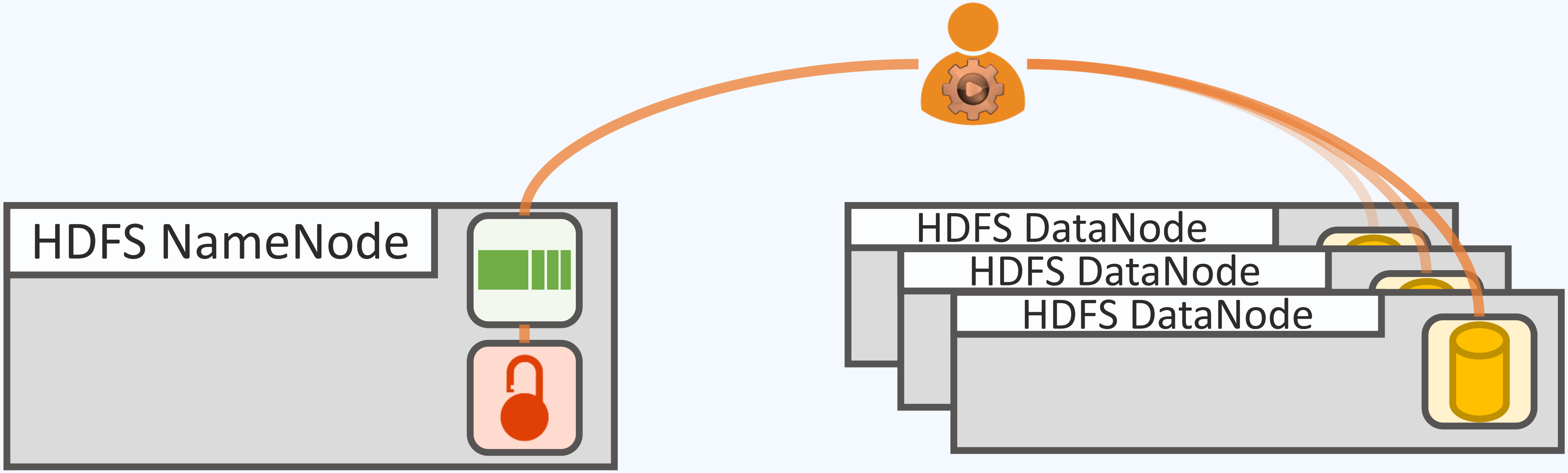
HDFS DataNode

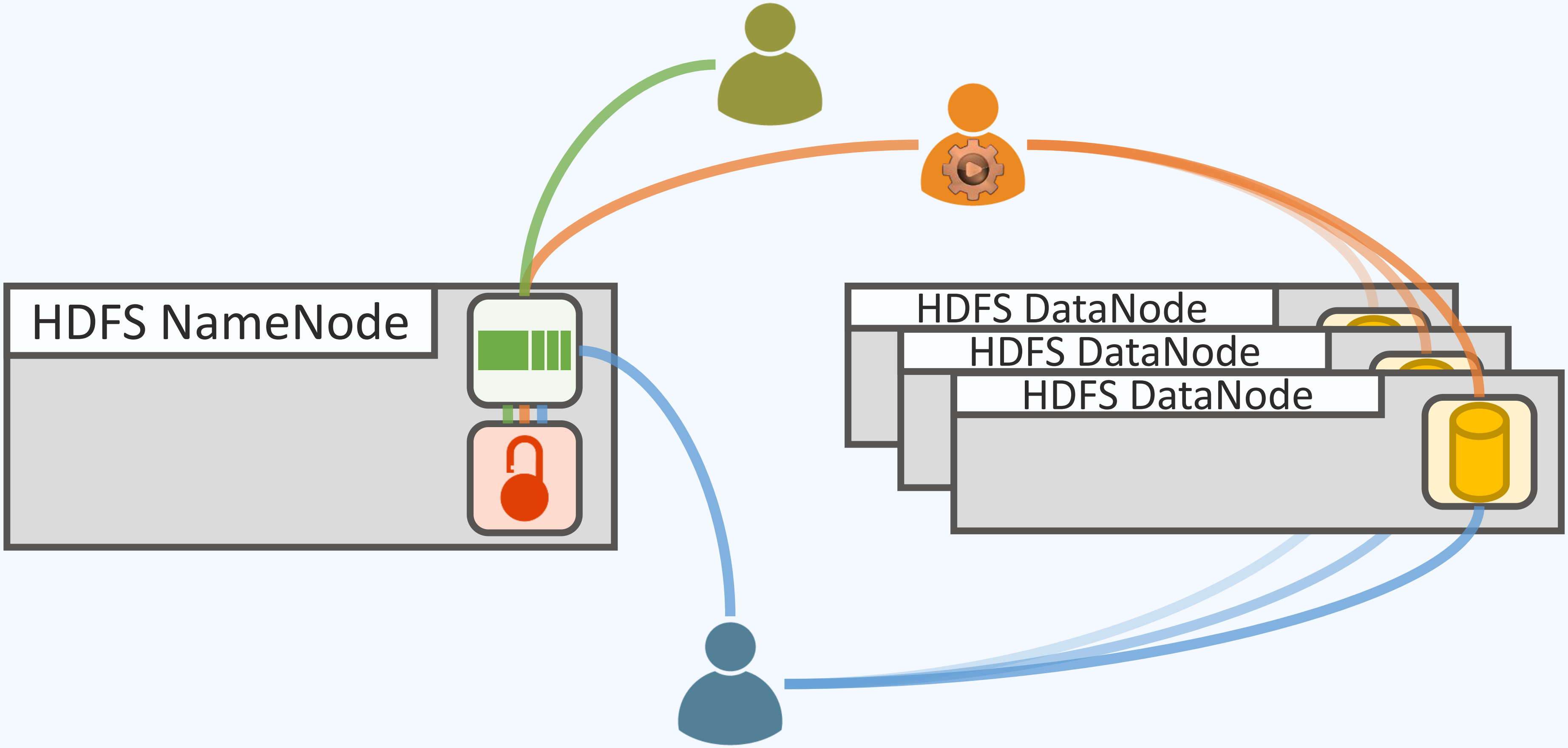
HDFS DataNode

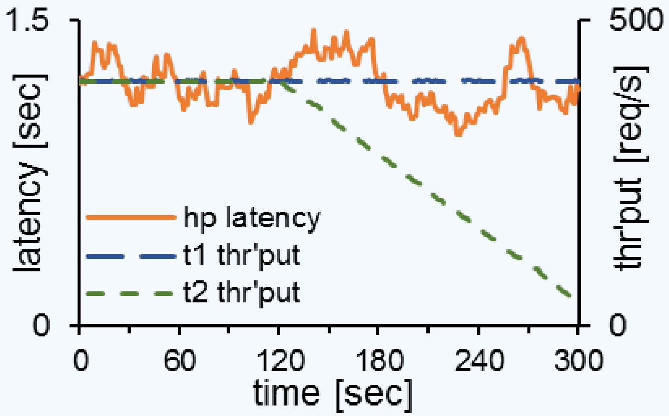
Principle 1: *Consider all resources and request types*

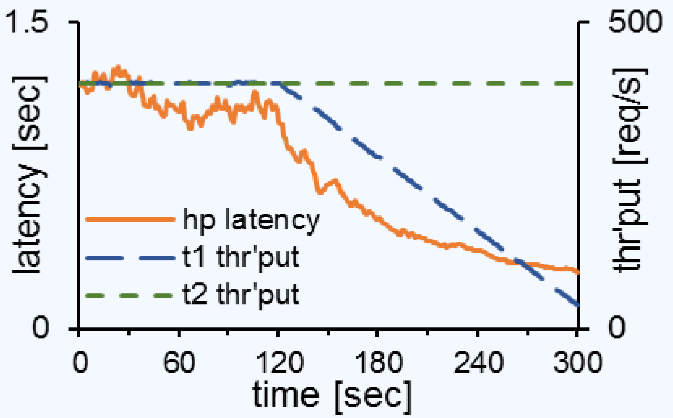
- Fine-grained resources within processes
- Resources shared between processes (disk, network)
- Many different API calls
- Bottlenecks can crop up in many places

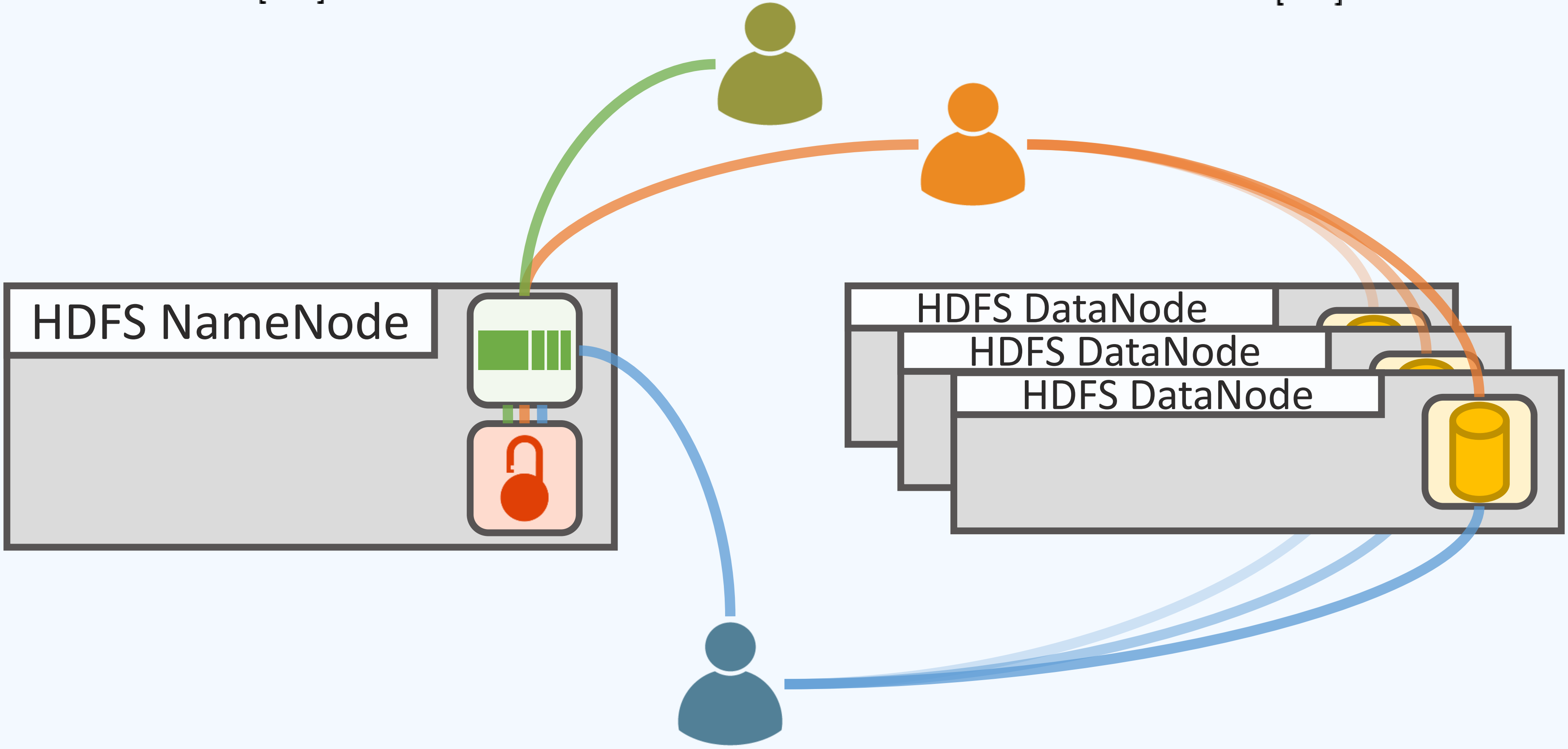
  hardware resources: *disk, network, cpu, …*
  software resources*: locks, queues, …*
  data structures: *transaction logs, shared batches, …*

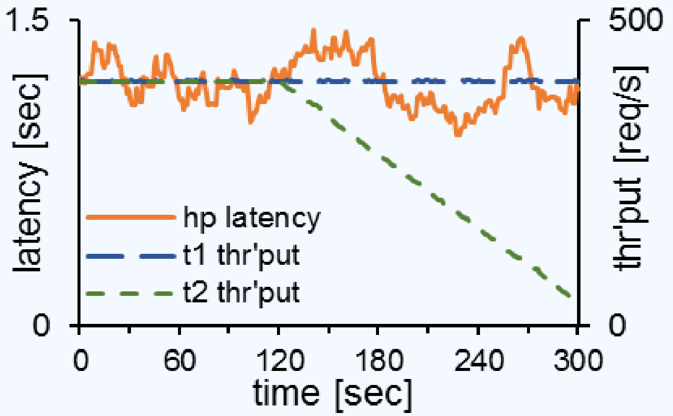

HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode

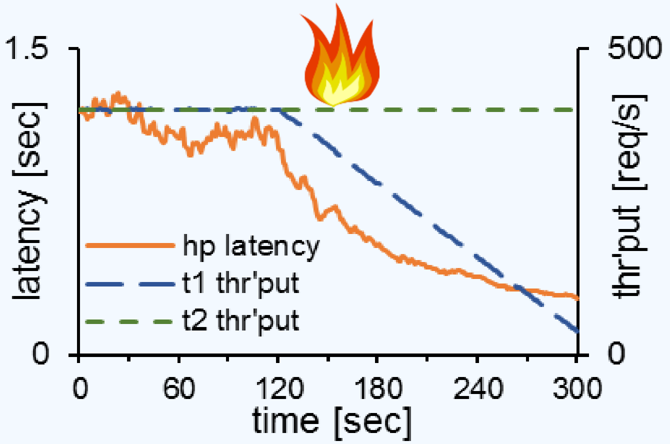

HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode

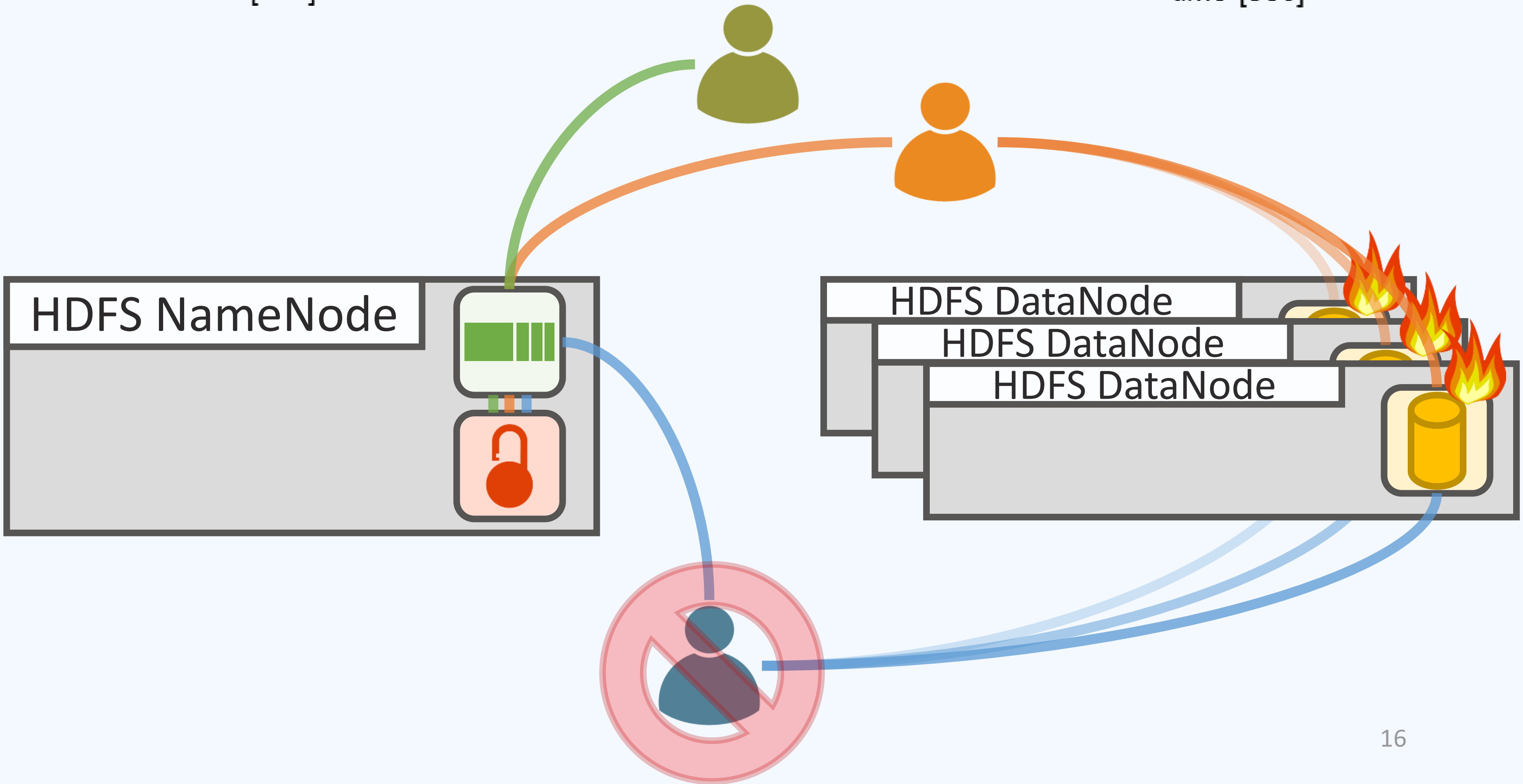

1.5
0
500
0
latency [sec]
thr'put [req/s]
hp latency
t1 thr'put
t2 thr'put
0
60
120
180
240
300
time [sec]


1.5
0
500
0
latency [sec]
thr'put [req/s]
hp latency
t1 thr'put
t2 thr'put
0
60
120
180
240
300
time [sec]


HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode

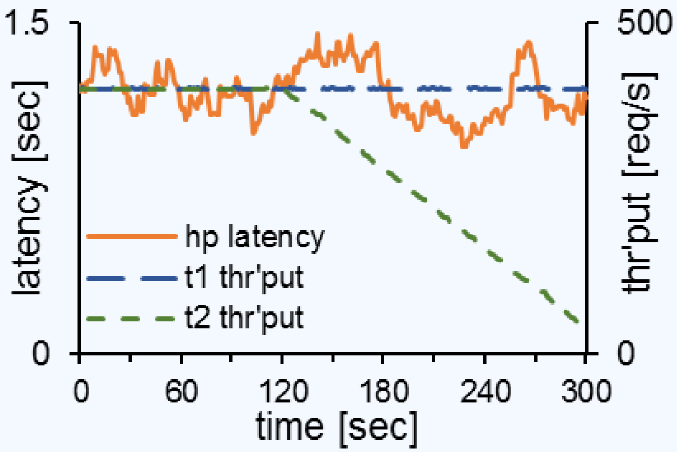

1.5
0
latency [sec]
500
0
thr'put [req/s]
0
60
120
180
240
300
time [sec]
hp latency
t1 thr'put
t2 thr'put


1.5
0
latency [sec]
500
0
thr'put [req/s]
0
60
120
180
240
300
time [sec]
hp latency
t1 thr'put
t2 thr'put


HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode

Principle 2: *Distinguish between tenants*

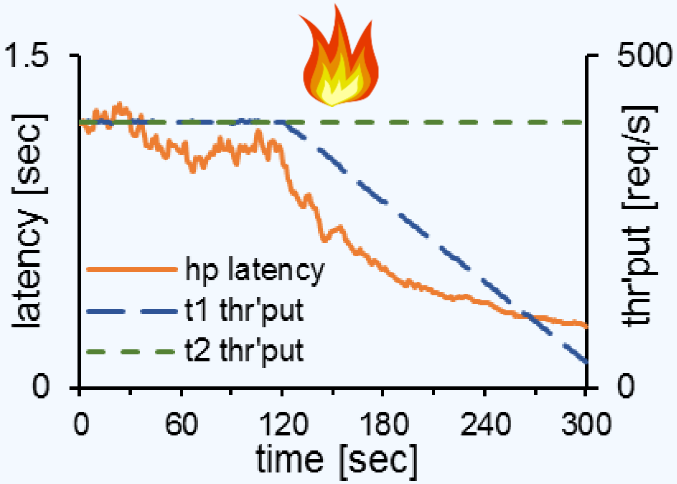
- Tenants might send different types of requests
- Tenants might be utilizing different machines

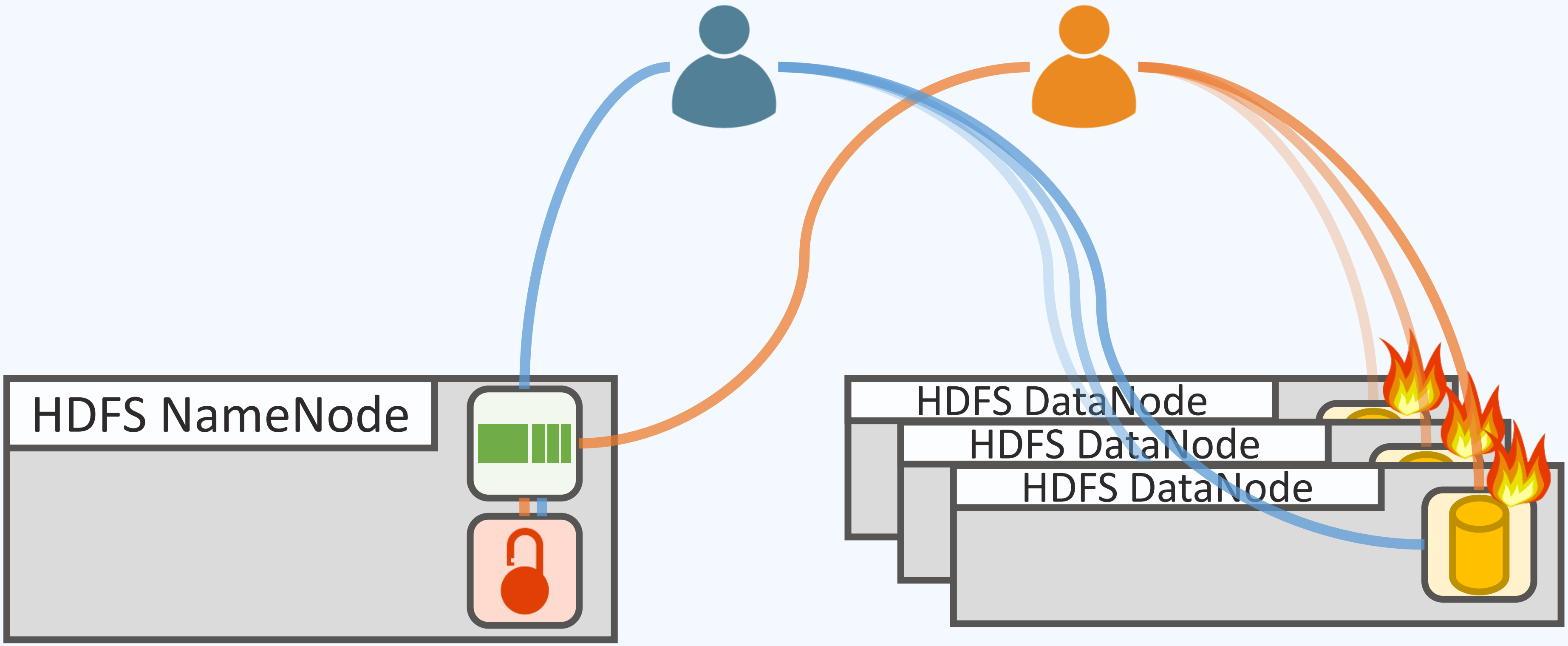


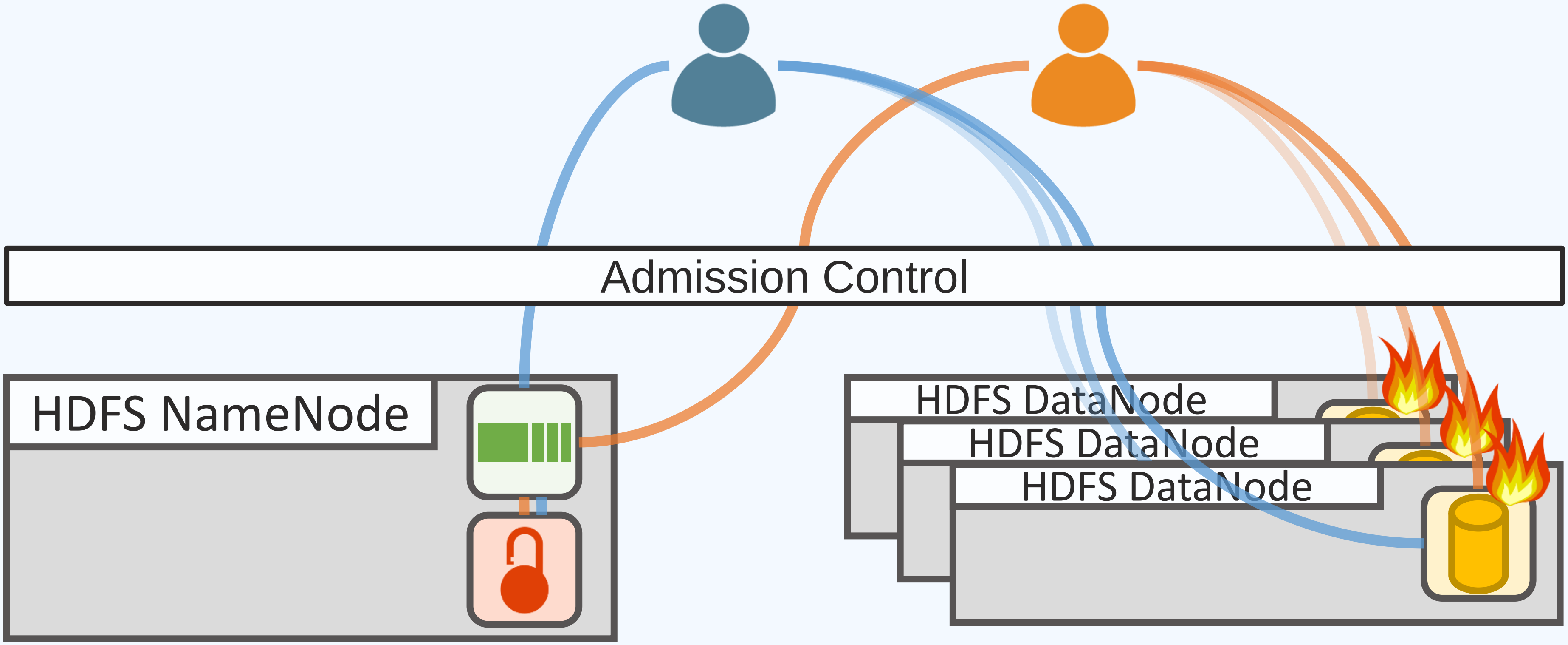
- If a policy is *efficient*, it should be able to *target* the cause of contention

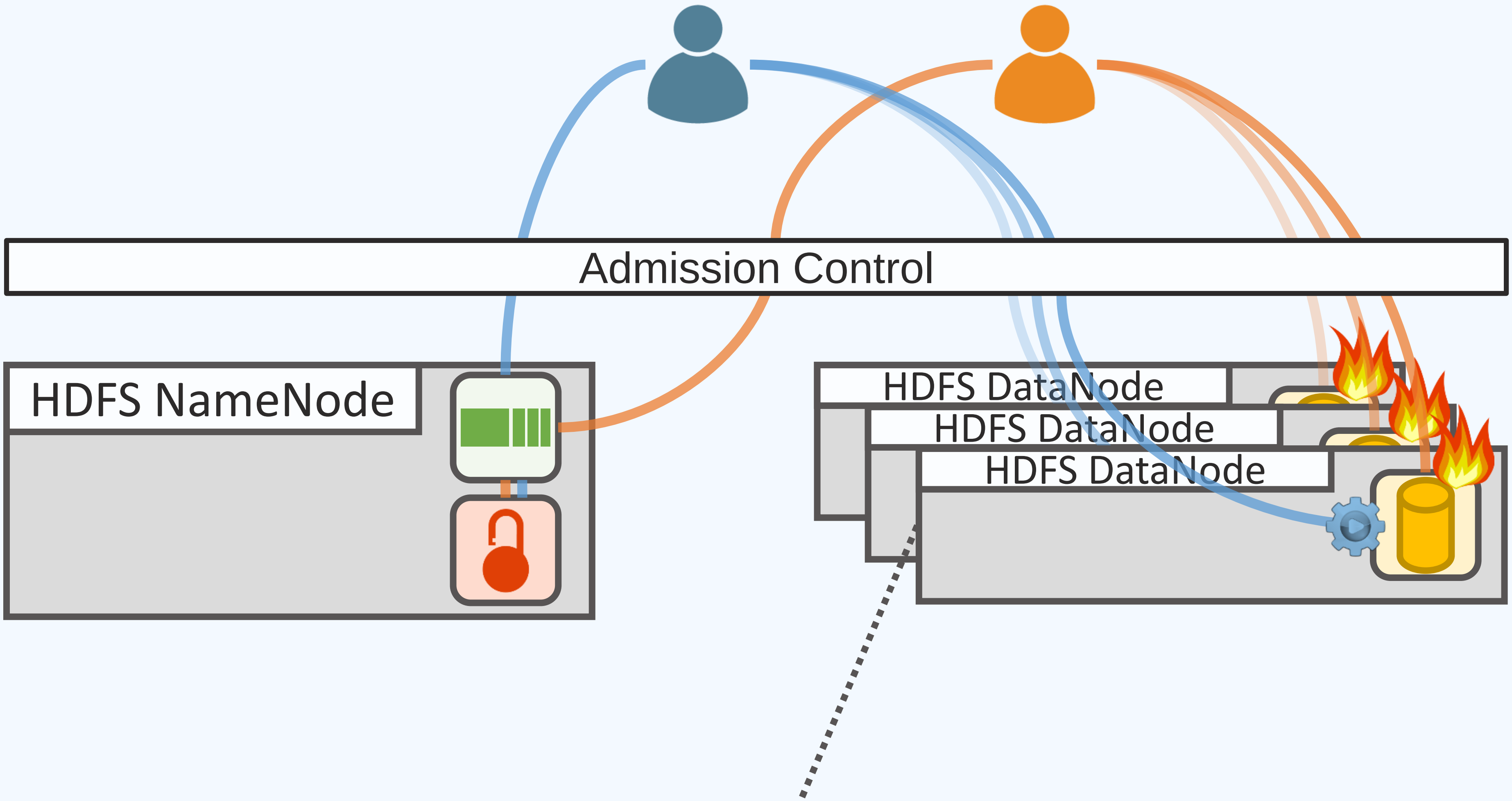
  e.g.,

  if a tenant is causing contention, throttle otherwise leave the tenant alone

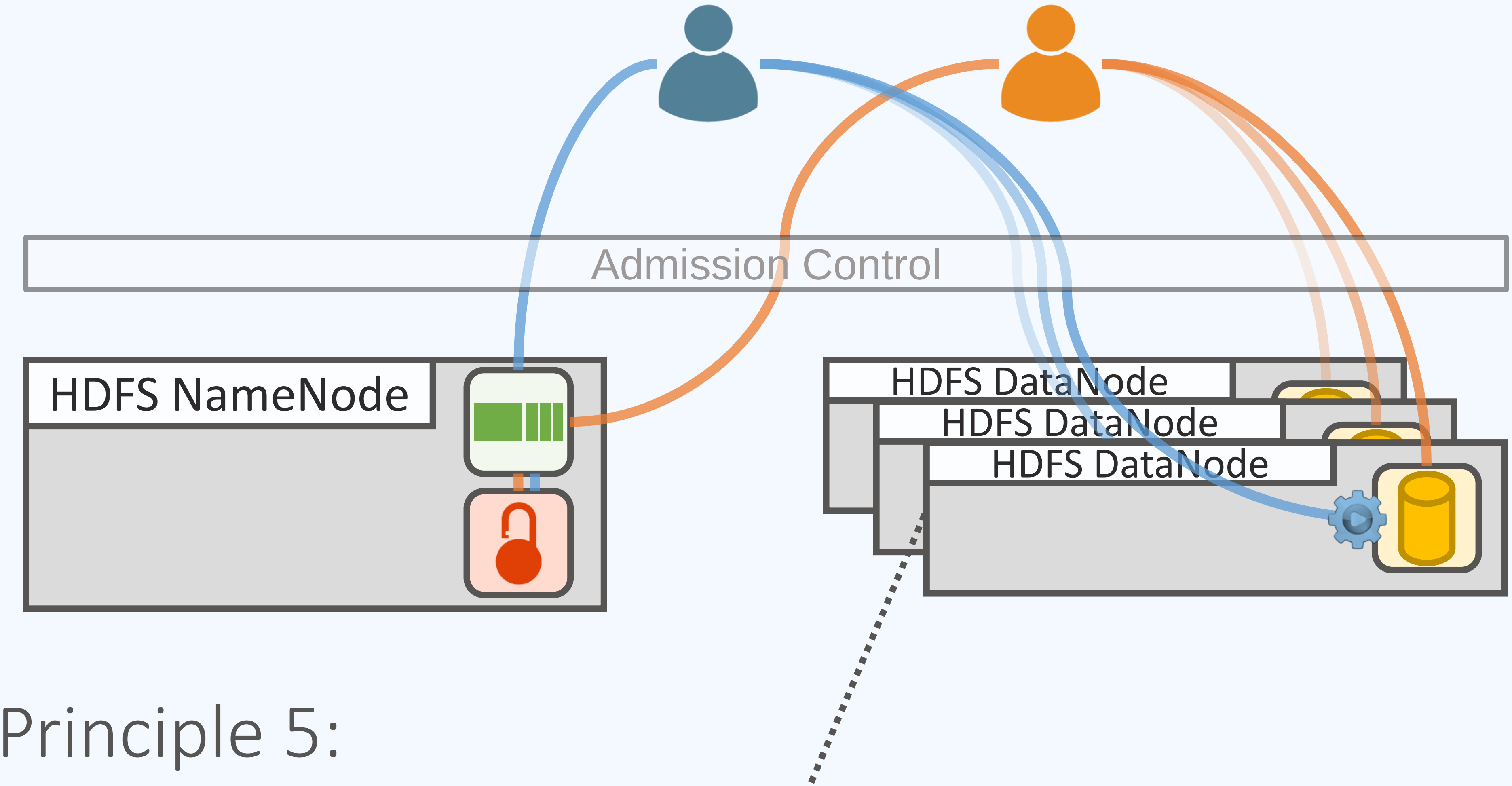



HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode


Admission Control
HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode



```
while (!Thread.isInterrupted()){
  sendPacket();
}
```



Principle 5:

Schedule early, schedule often

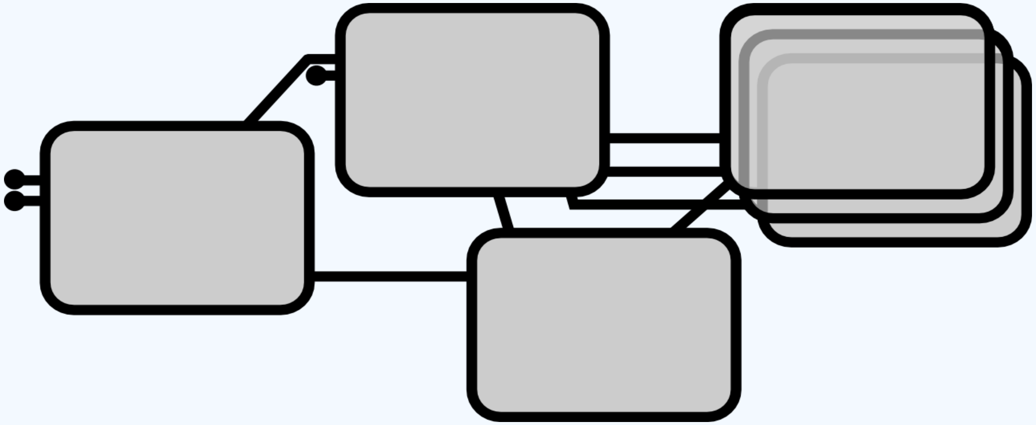
```
while (!Thread.isInterrupted()){
    rate_limit();
    sendPacket();
}
```

Resource Management Design Principles

1. **Consider all request types and all resources**
2. **Distinguish between tenants**
3. Treat foreground and background tasks uniformly
4. Estimate resource usage at runtime
5. **Schedule early, schedule often**

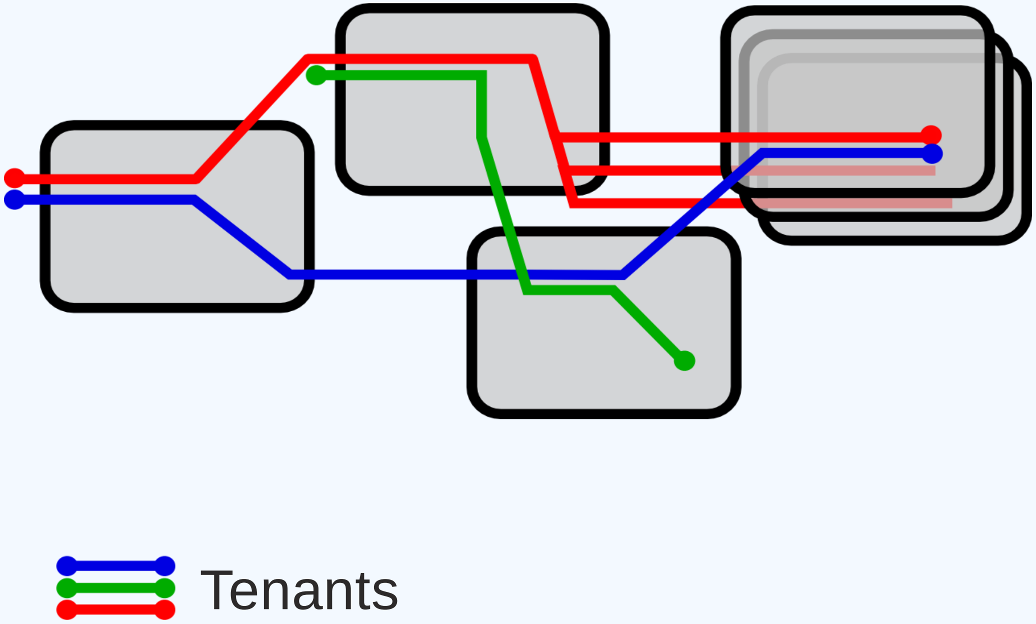
Retro – prototype for **principled** resource management in shared-tenant systems

Retro: end-to-end tracing



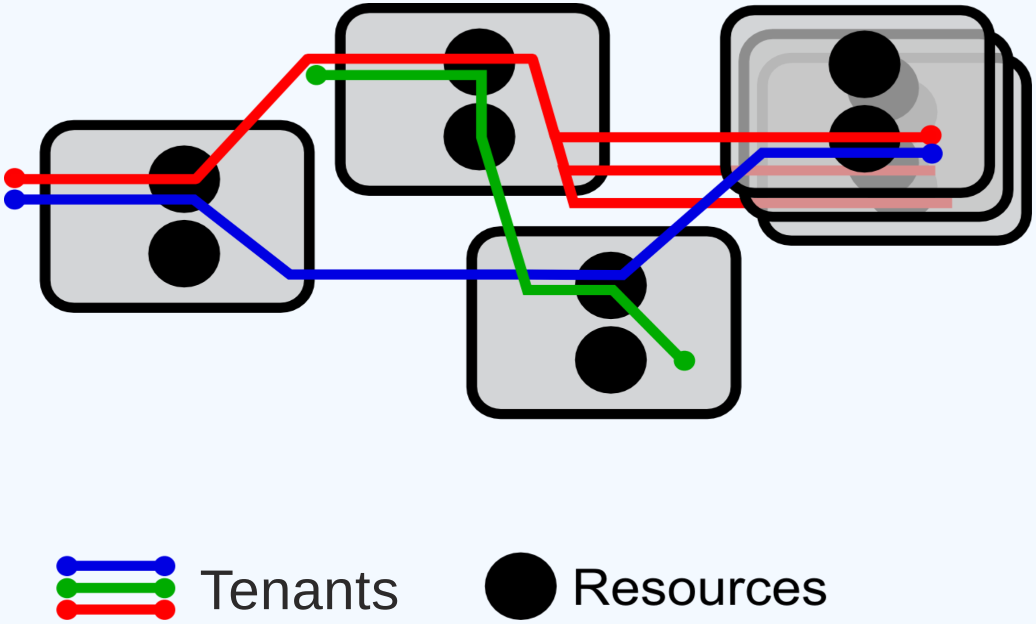
Tenants

Retro: end-to-end tracing

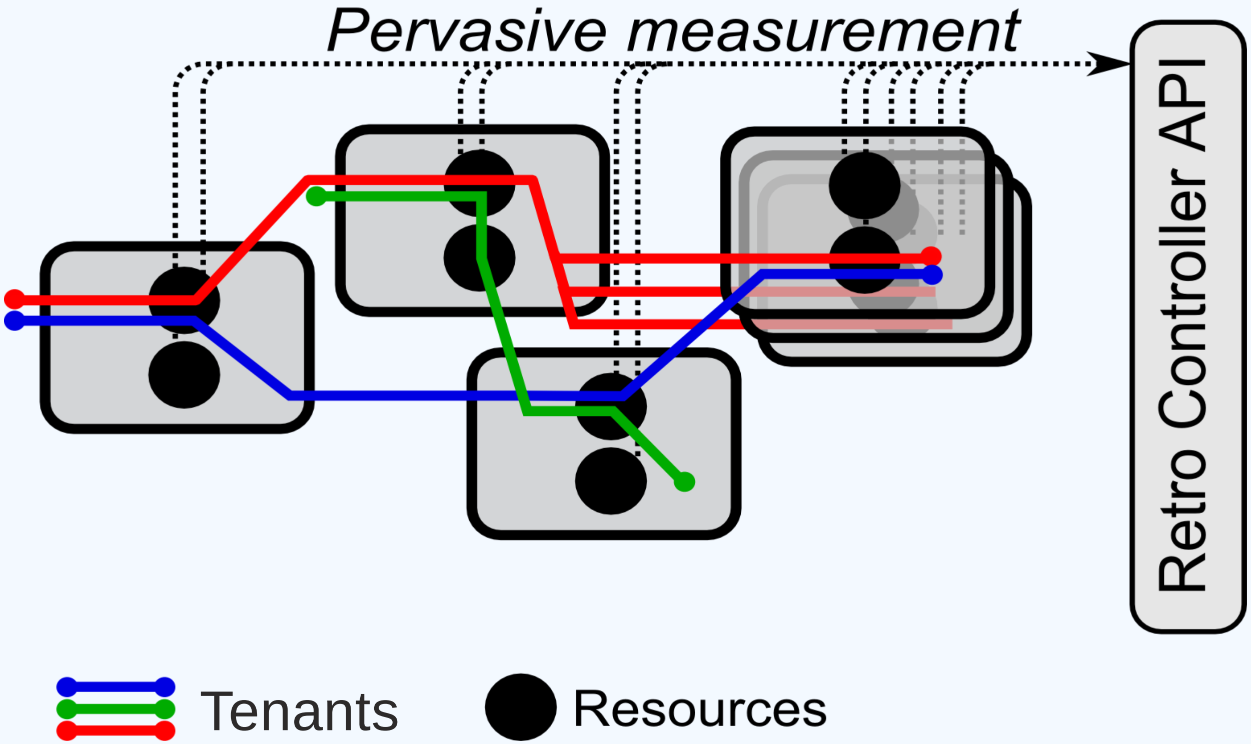


Tenants

Retro: application-level resource interception



Retro: aggregation and centralized reporting



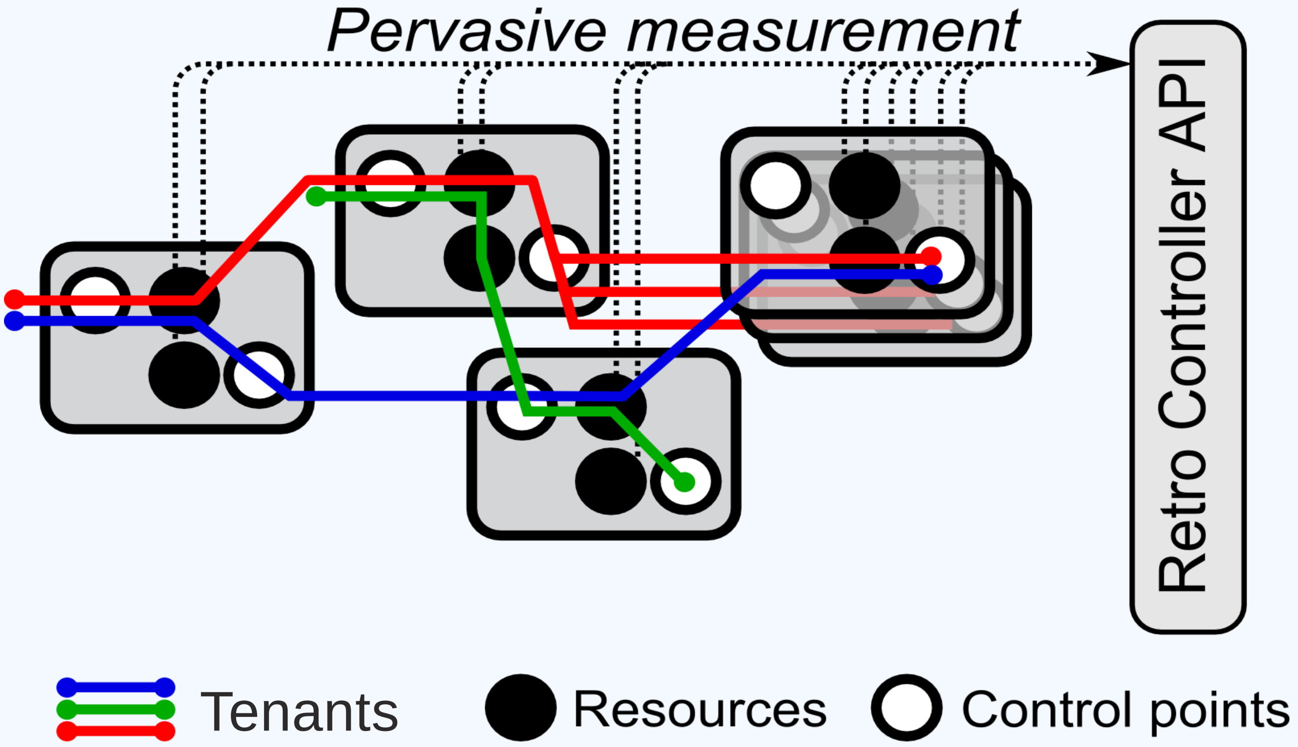
Pervasive measurement

Retro Controller API

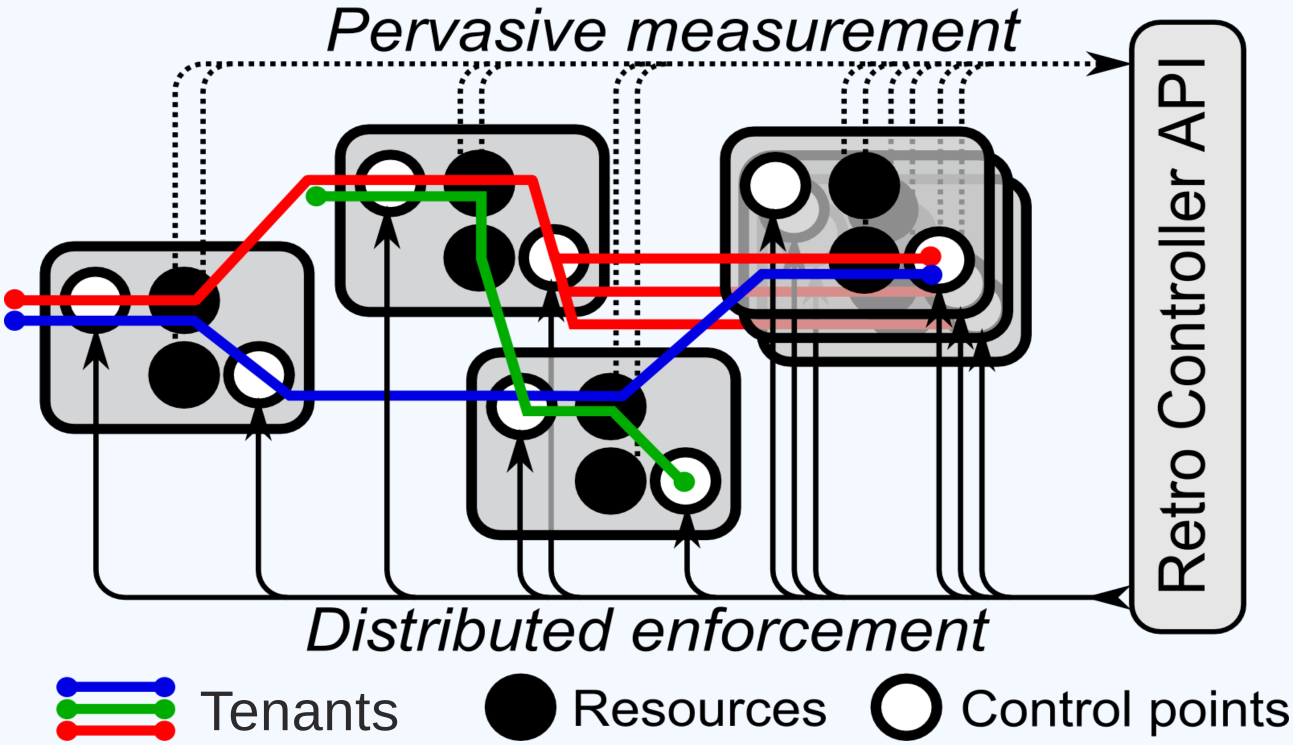
Tenants

Resources

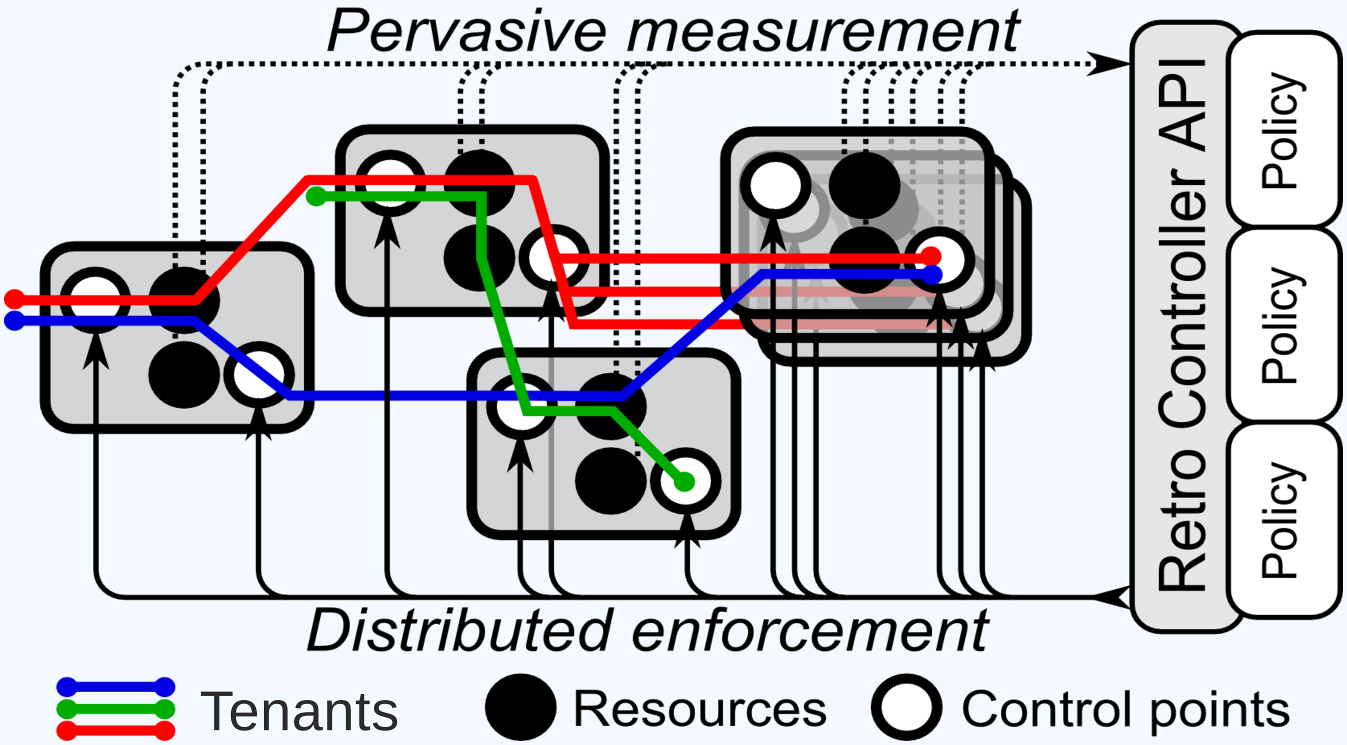
Retro: application-level enforcement



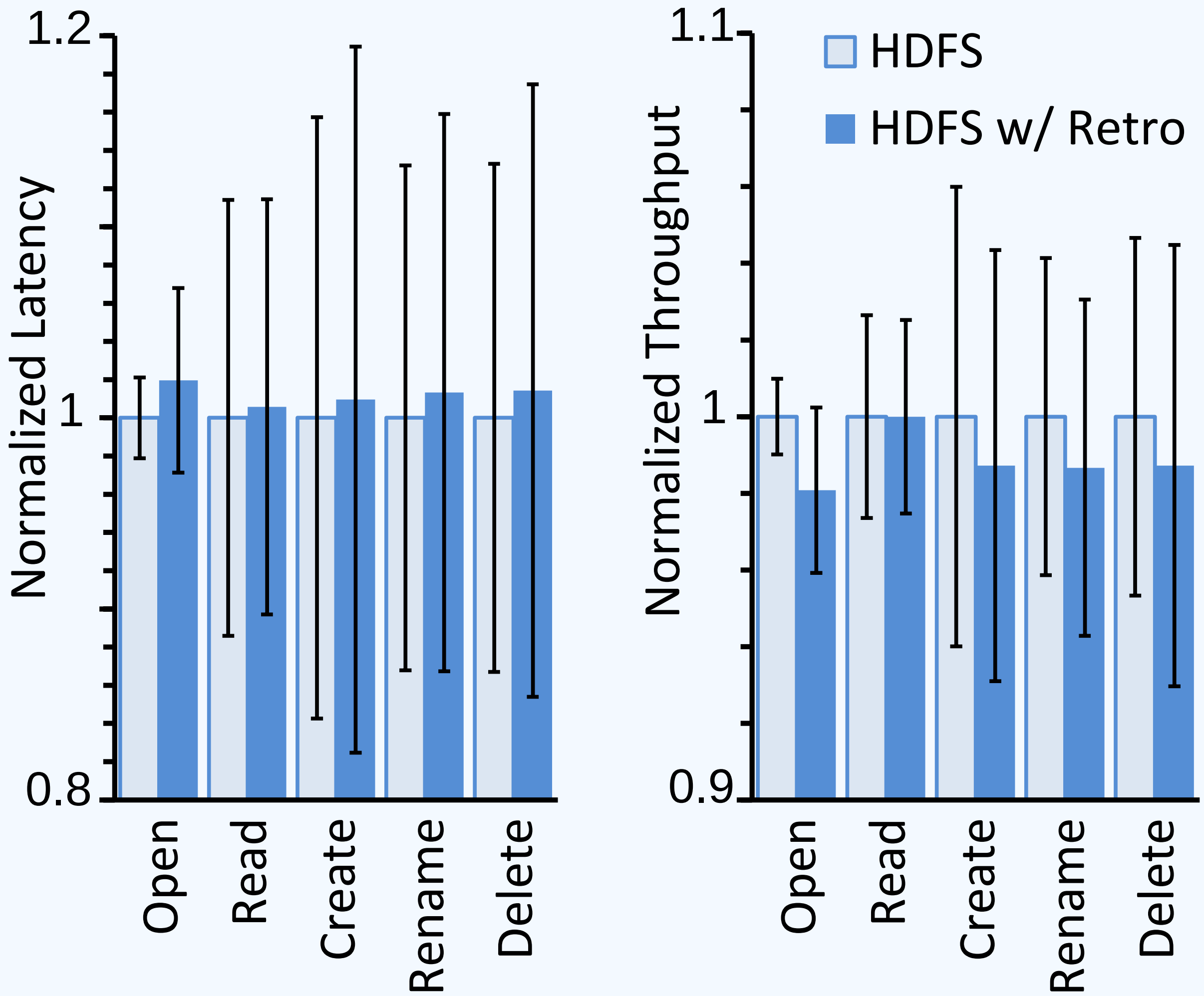
Retro: distributed scheduling



Retro: distributed scheduling

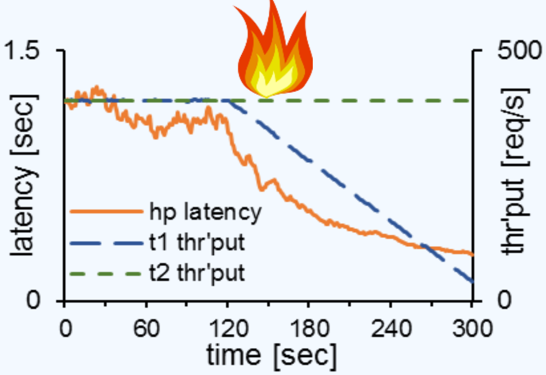


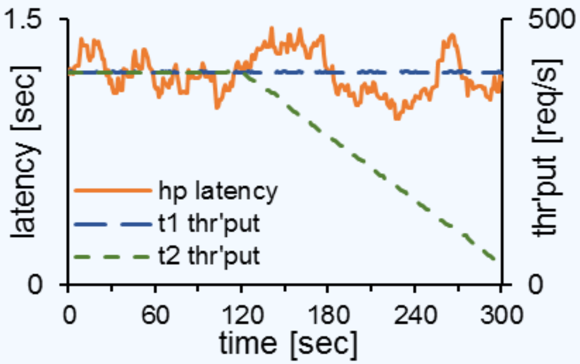
Early Results

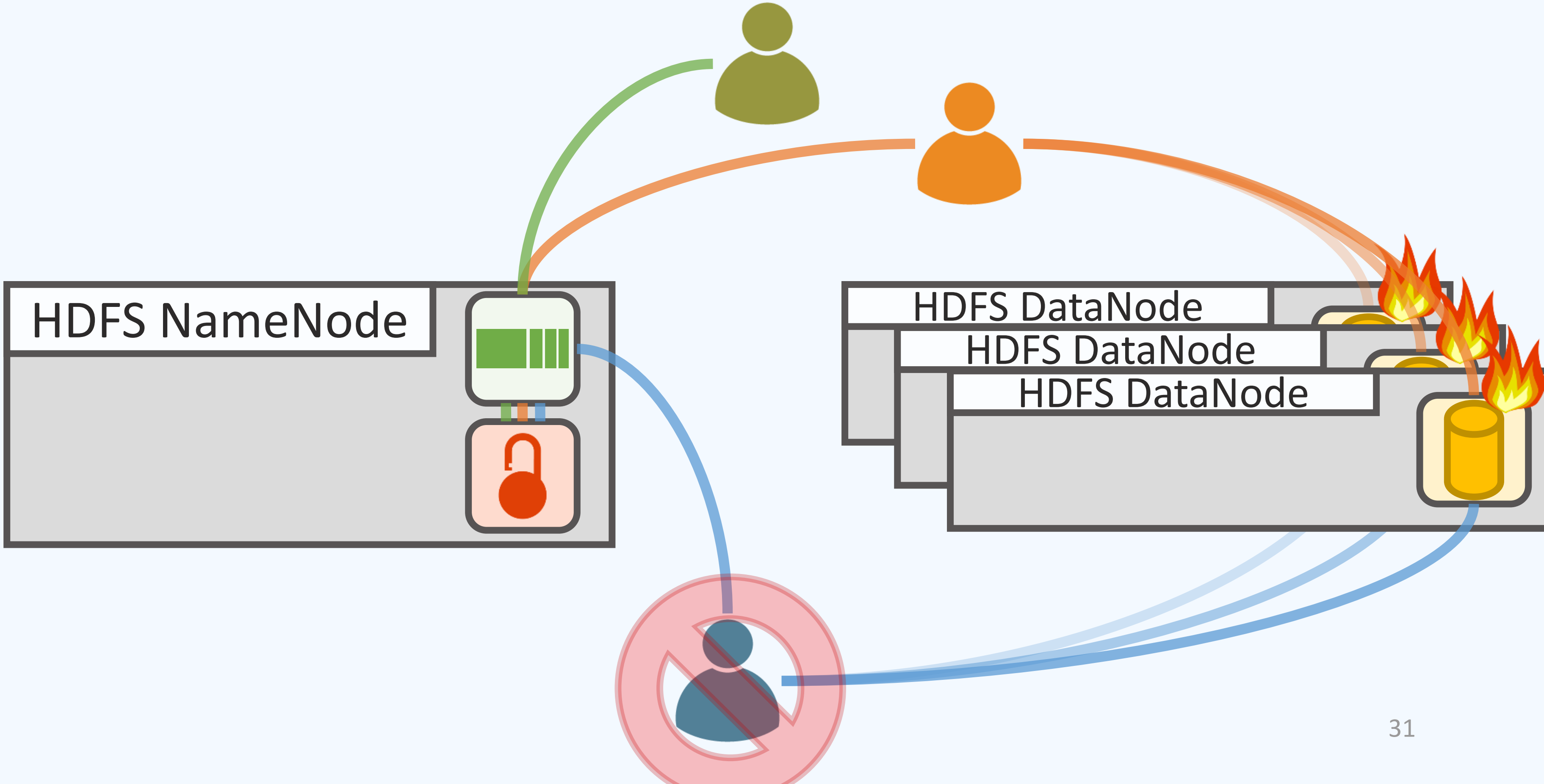


HDFS NNBench benchmark

0.01% to 2% average overhead on end-to-end latency, throughput





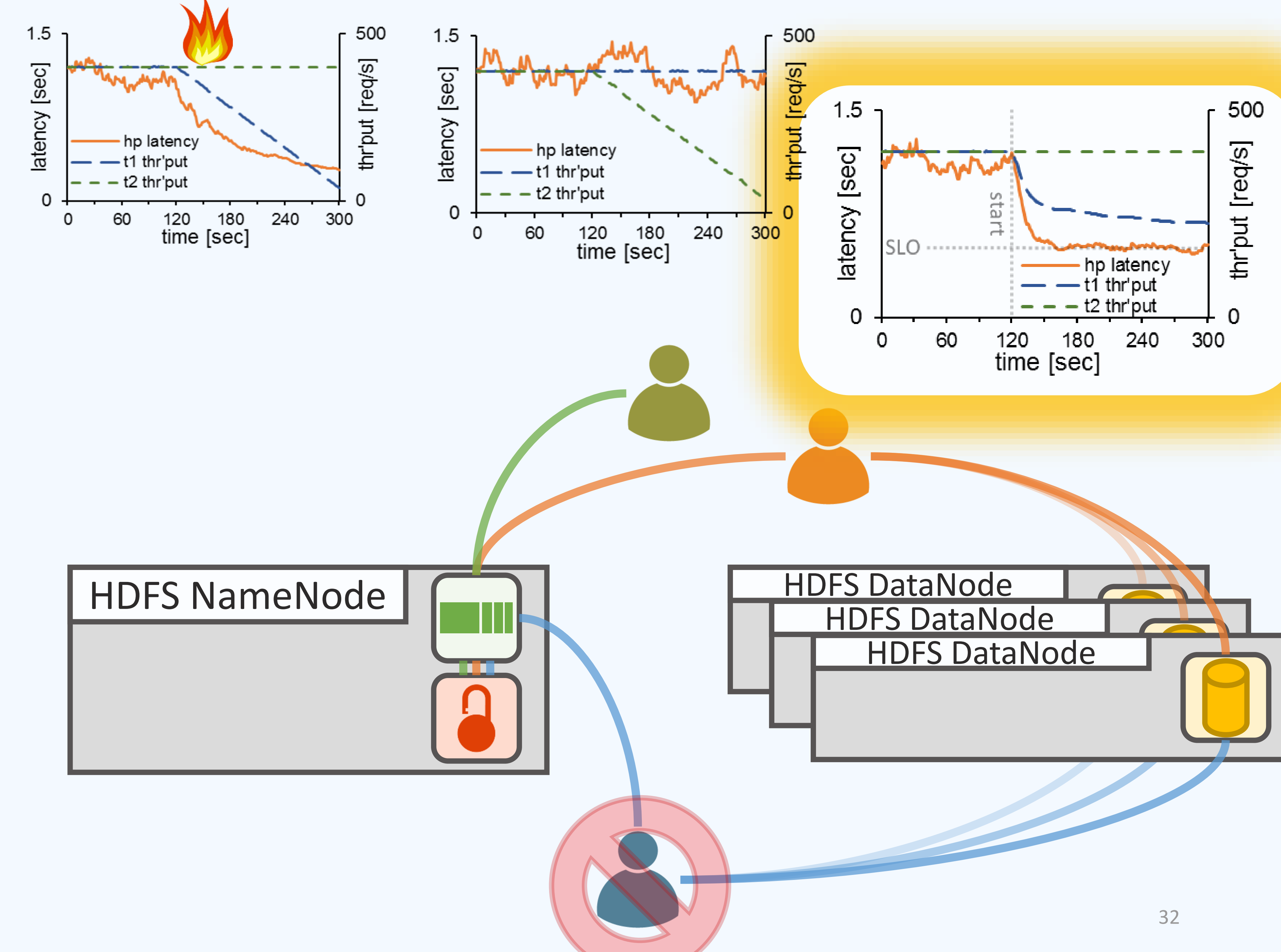


HDFS NameNode

HDFS DataNode

HDFS DataNode

HDFS DataNode


1.5
0
latency [sec]
500
0
thr'put [req/s]
0
60
120
180
240
300
time [sec]
hp latency
t1 thr'put
t2 thr'put
1.5
0
latency [sec]
500
0
thr'put [req/s]
0
60
120
180
240
300
time [sec]
hp latency
t1 thr'put
t2 thr'put
1.5
0
latency [sec]
500
0
thr'put [req/s]
start
SLO
0
60
120
180
240
300
time [sec]
hp latency
t1 thr'put
t2 thr'put
HDFS NameNode
HDFS DataNode
HDFS DataNode
HDFS DataNode

Retrospective

Thus far:

- Per-tenant identification
- Resource measurements
- Schedule enforcement

Next steps:

- Abstractions for writing simplified high-level policies
- Low-level enforcement mechanisms
- Policies to monitor system, find bottlenecks, provide guarantees