



Choosing Hardware For MySQL

Kenny Gryp <kenny.gryp@percona.com>
Percona Live Washington DC / 2012-01-11

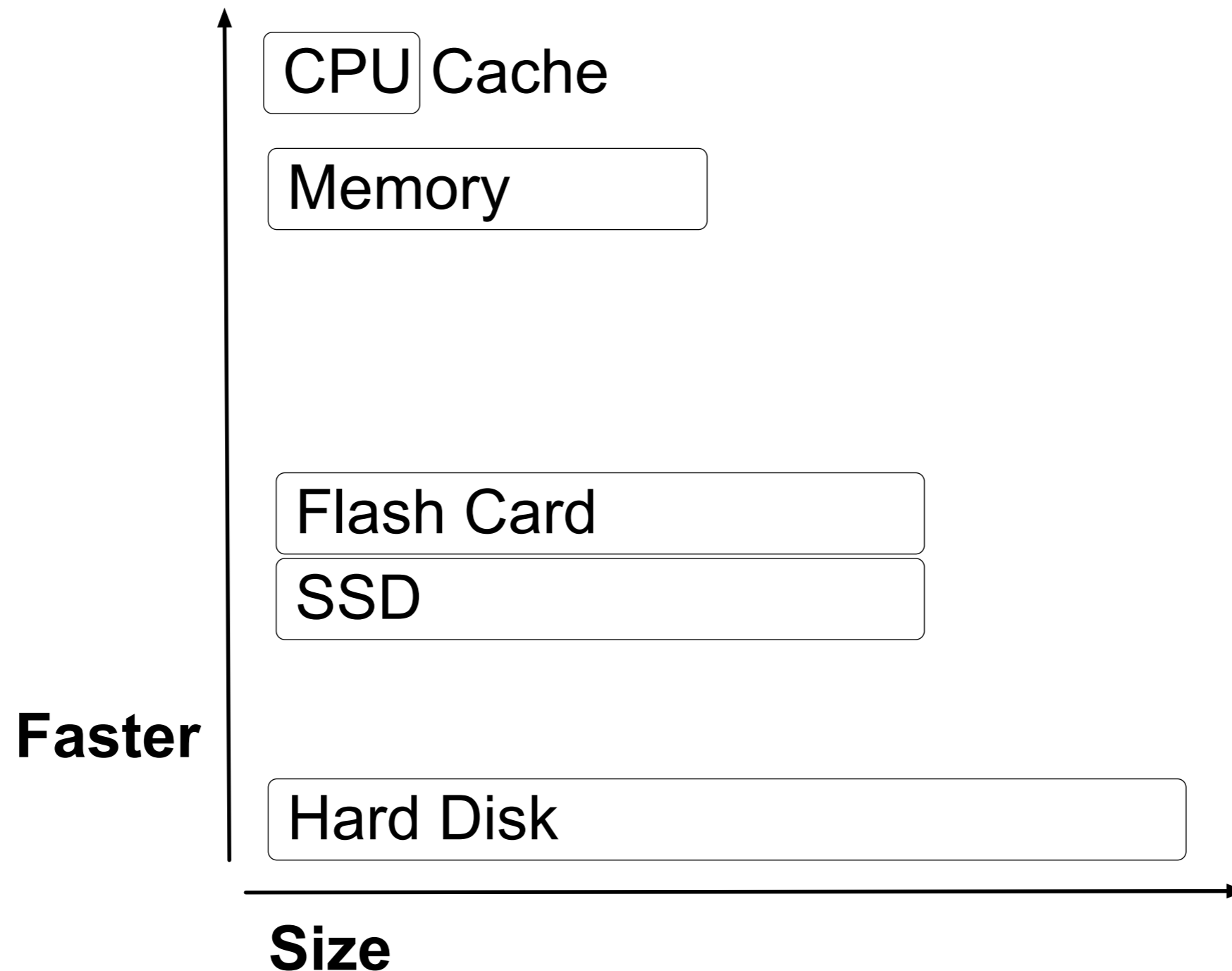
Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL

Choosing Hardware For MySQL

- ◆ **Numbers Everybody Should Know**
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL

Numbers Everybody Should Know



Numbers Everybody Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

See: <http://www.linux-mag.com/cache/7589/1.html> and

Google <http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf>

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ **CPU**
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL

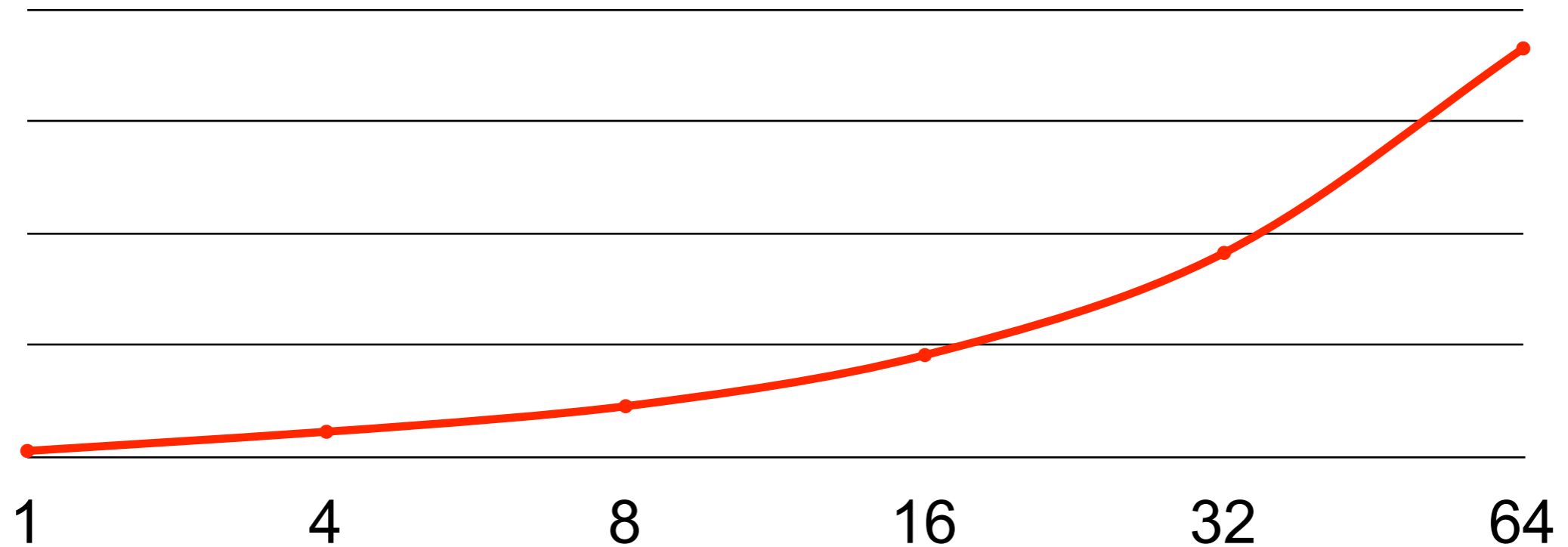
CPU

- ◆ Scalability
- ◆ Mutex Contention
- ◆ Response Time & Throughput
- ◆ When are CPU's the problem?
- ◆ CPUs
- ◆ CPU Utilization

Scalability

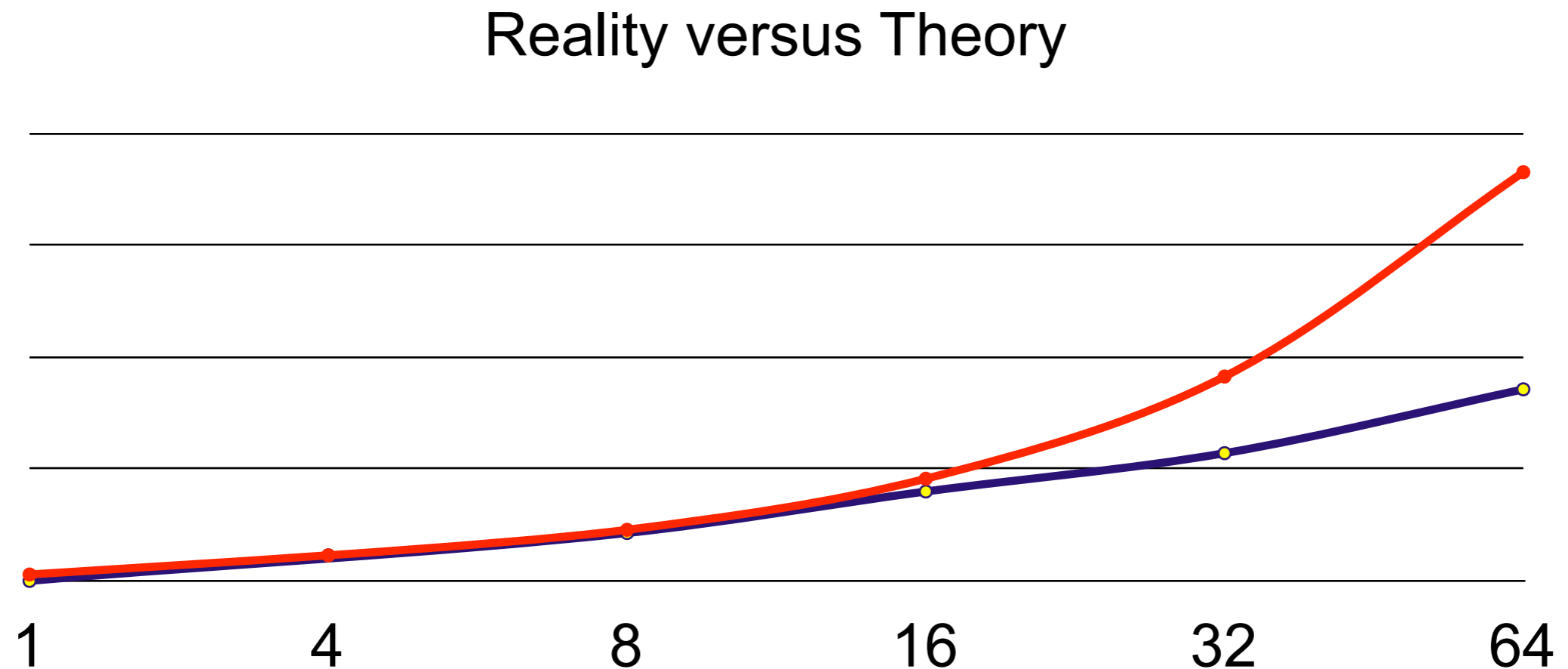
- ◆ **[Perfect World]:** As we add CPUs we get a linear throughput increase, provided we have sufficient concurrency:

Theoretical Workload



CPU Scalability

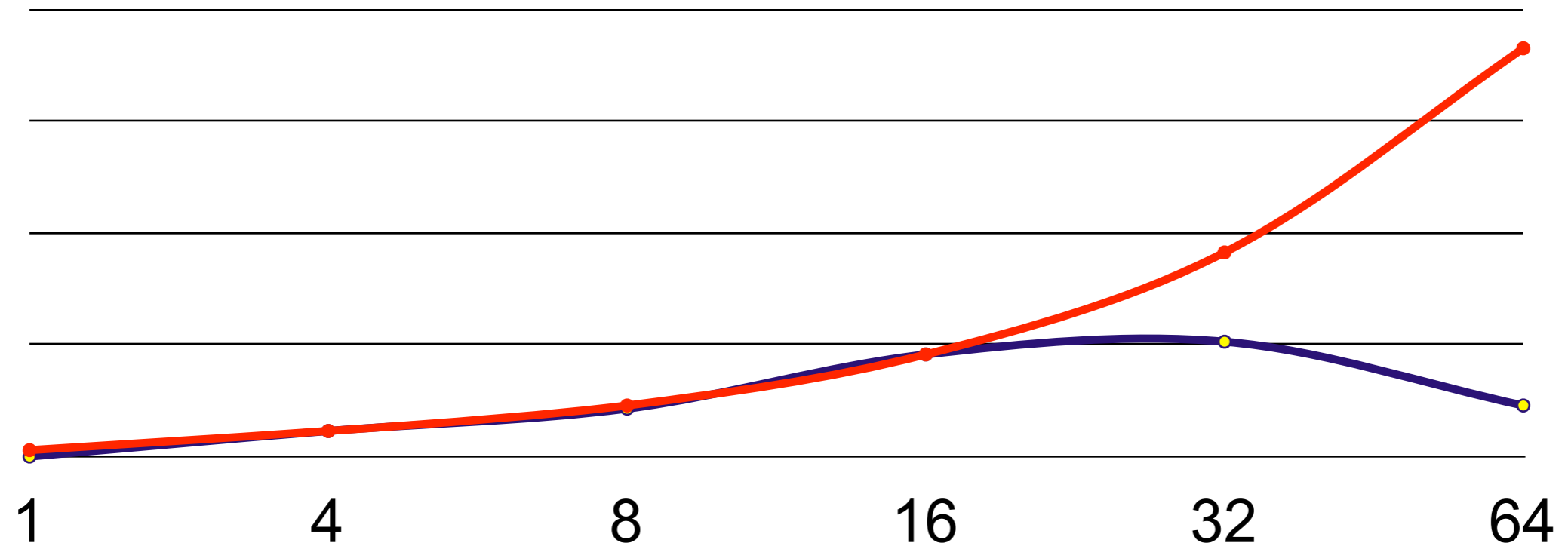
- ◆ **[Reality]:** We never quite follow the theoretical curve:



CPU Scalability

- ◆ **[Reality]:** Even this happens:

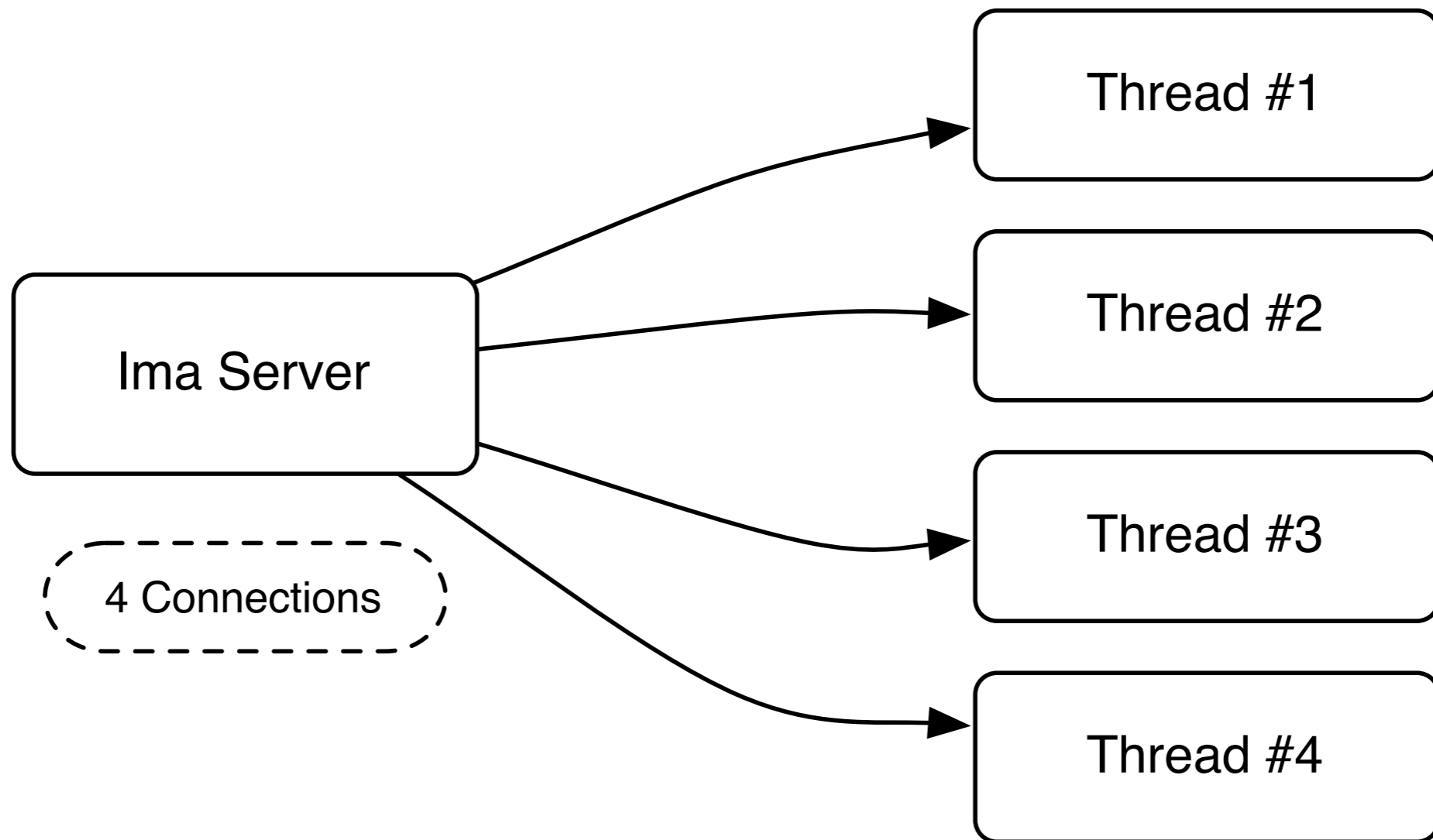
Reality versus Theory



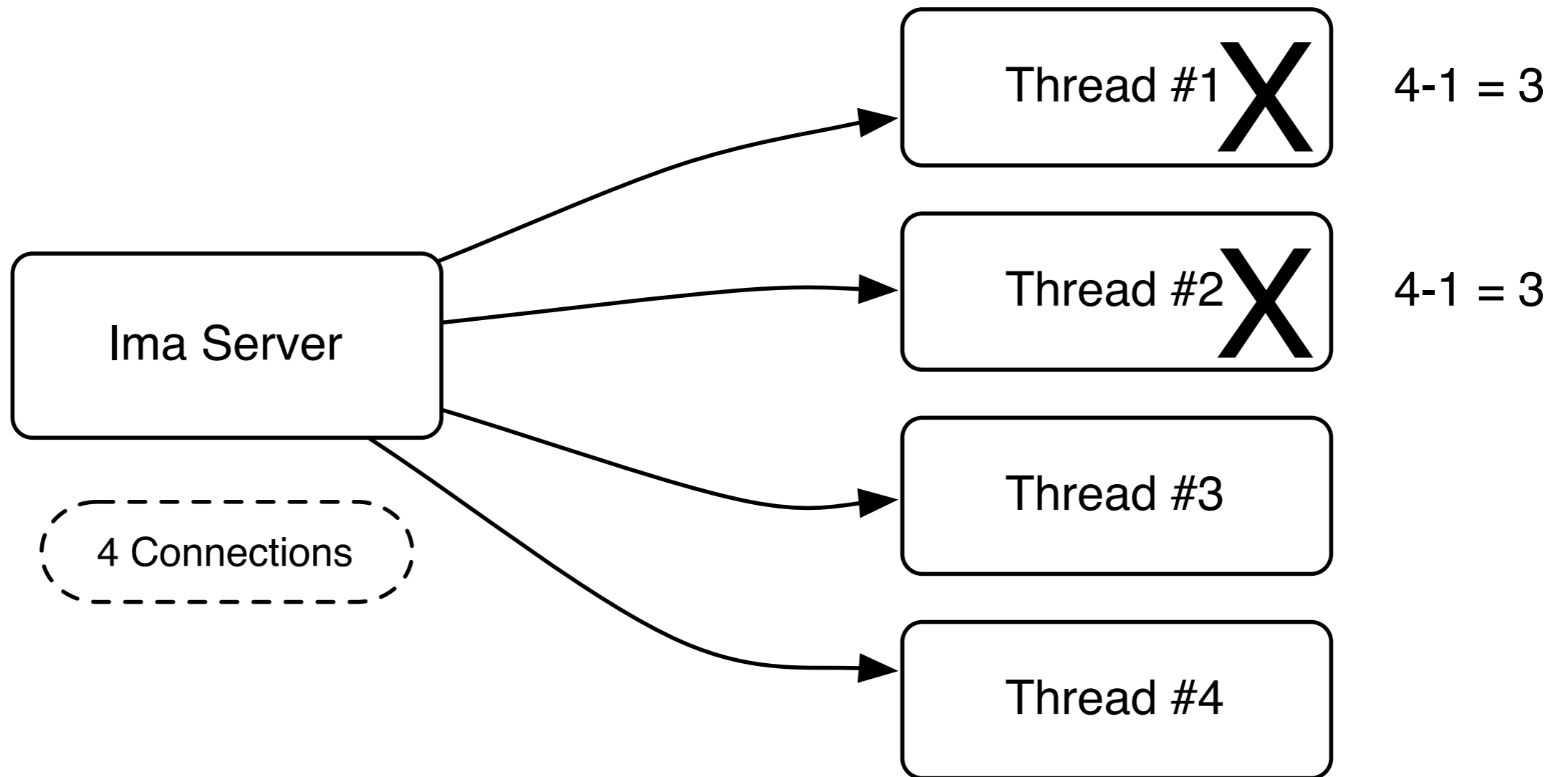
CPU Scalability

- ◆ Internal Contention: MySQL&InnoDB: Mutex Contention:
 - ◆ `kernel_mutex`, `rollback segment`, `buffer pool operations...`
- ◆ Logical Contention:
 - ◆ visible to the application
 - ◆ table/row level locking

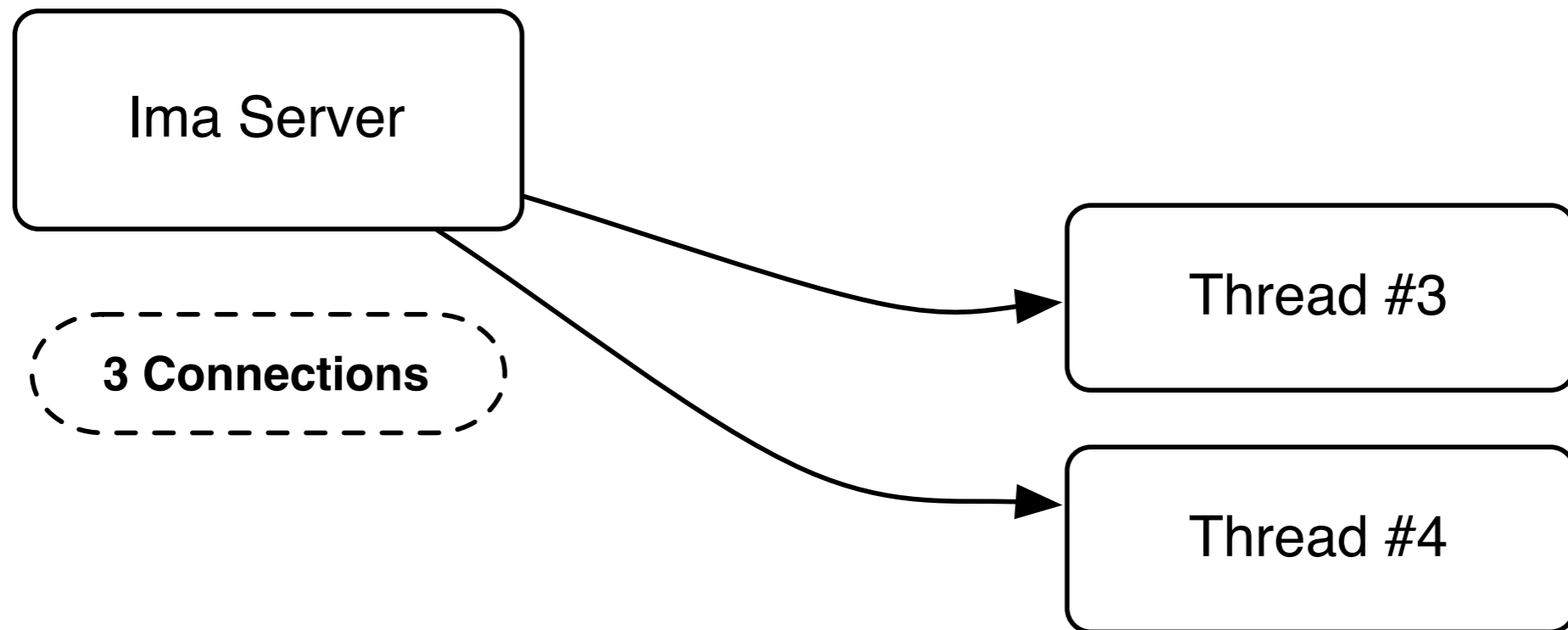
What's a Mutex?



What's a Mutex? (cont.)

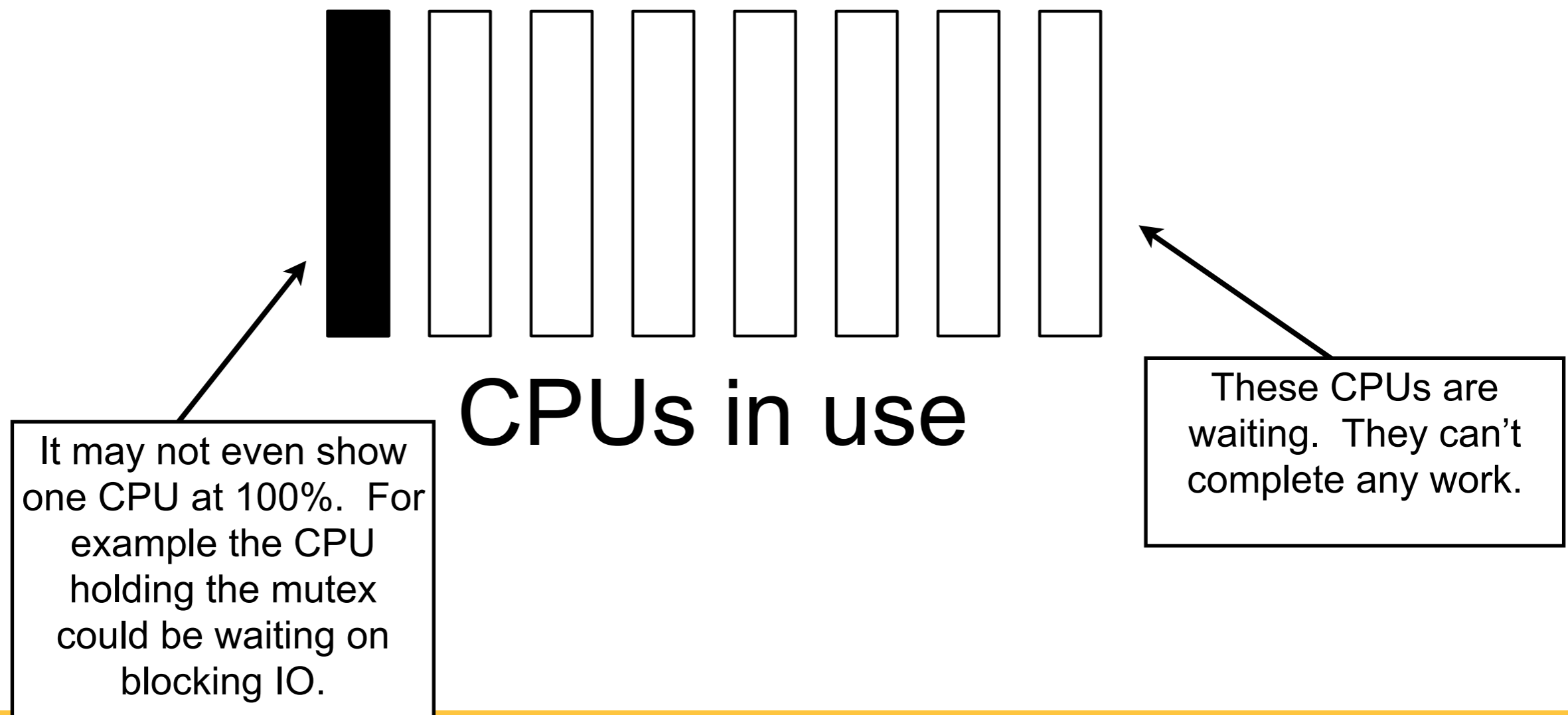


What's a Mutex? (cont.)



Mutexes Become Hotspots

- ♦ The longer the mutex is held, the more likely you can hold up other tasks - and reduce CPU scalability:



Mutex Contention in MySQL/InnoDB

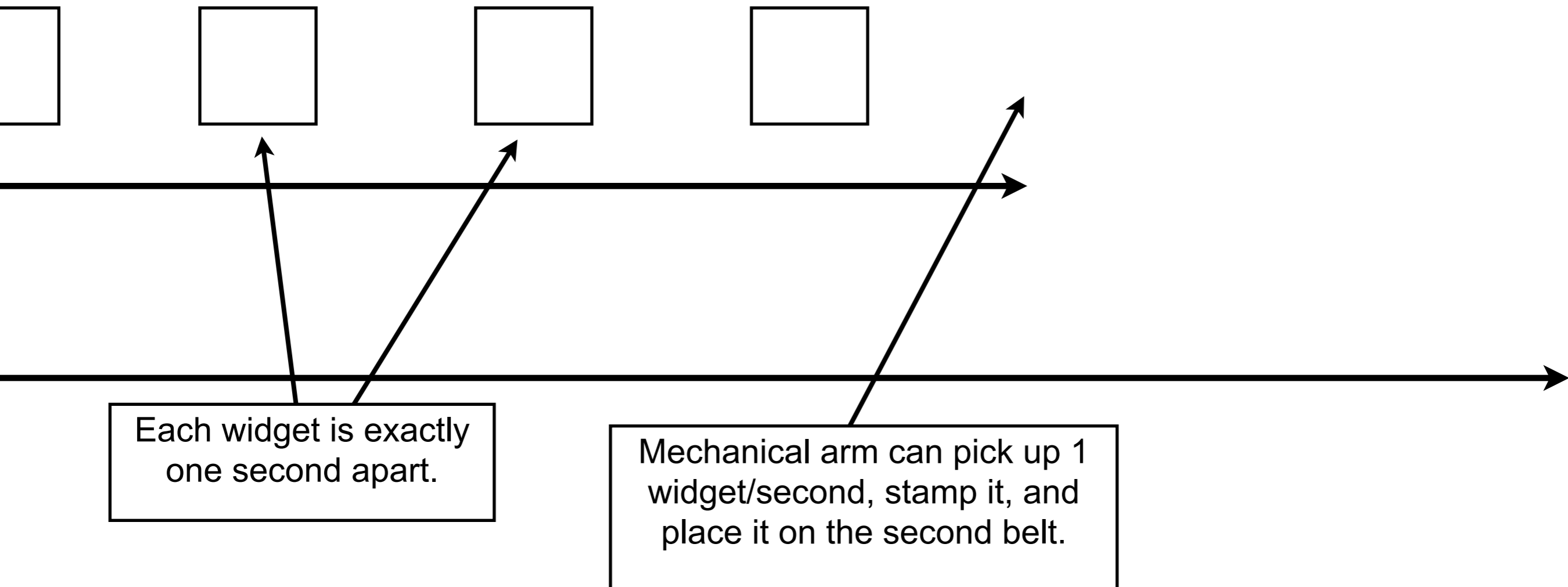
- ◆ In MySQL 5.0 (built-in InnoDB), a lot of mutex contention
- ◆ Since InnoDB Plugin 1.0 (5.1.x, not enabled by default), better scalability
- ◆ Many more enhancements in Percona Server XtraDB and InnoDB Plugin 1.1 (5.5)
- ◆ Still known Mutex contention places (eg. `kernel_mutex`) in 5.5, many more enhancements in 5.6

Performance

- ◆ Response Time = Time to run a Single Statement
- ◆ Throughput = Statements per Second
- ◆ What does your application need?

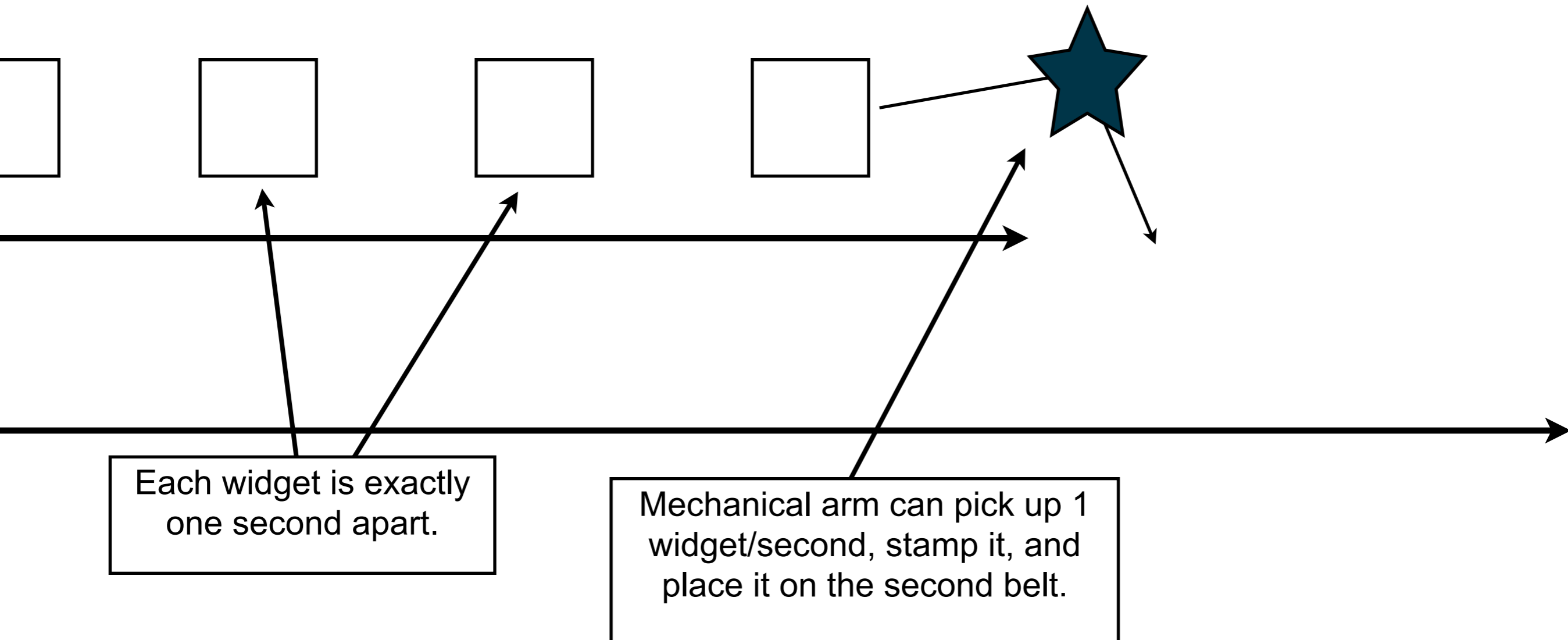
Next thing to know about CPUs

- ◆ Not all tasks arrive on time. Take the following example of a manufacturing process:



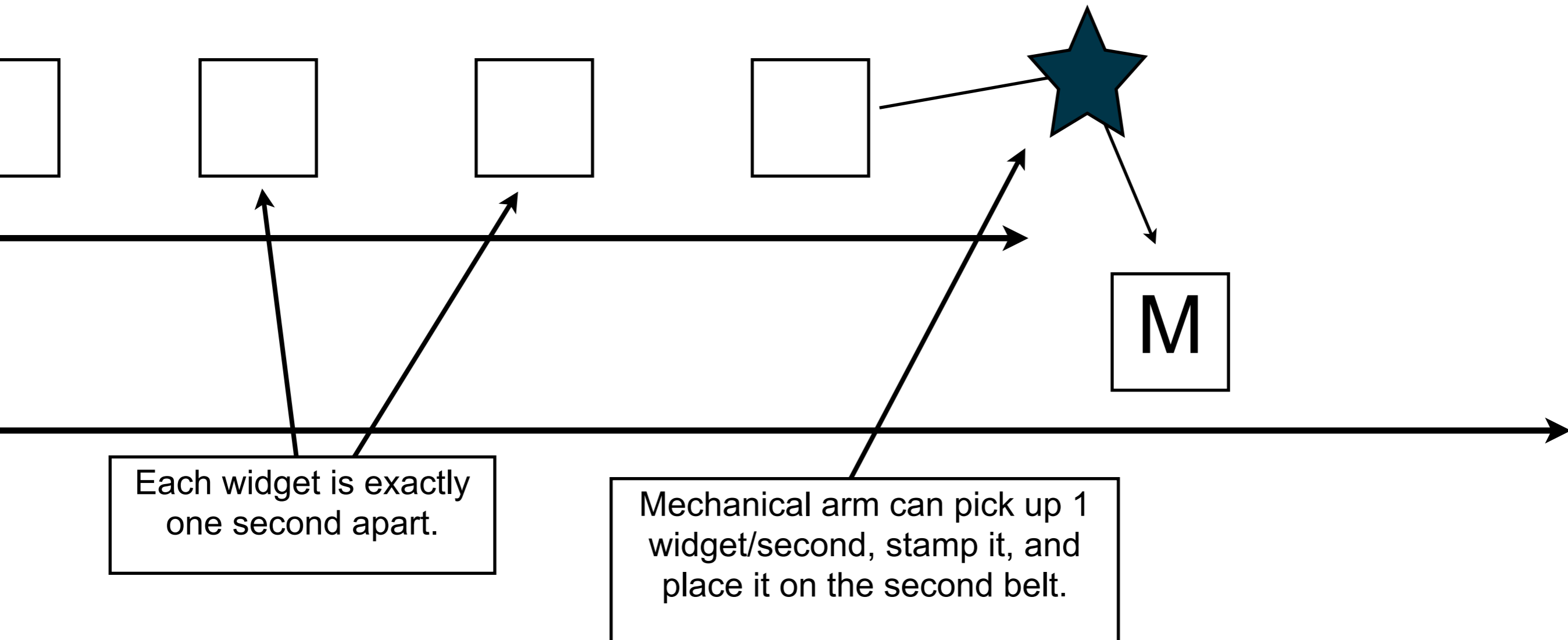
Next thing to know about CPUs

- ◆ Not all tasks arrive on time. Take the following example of a manufacturing process:



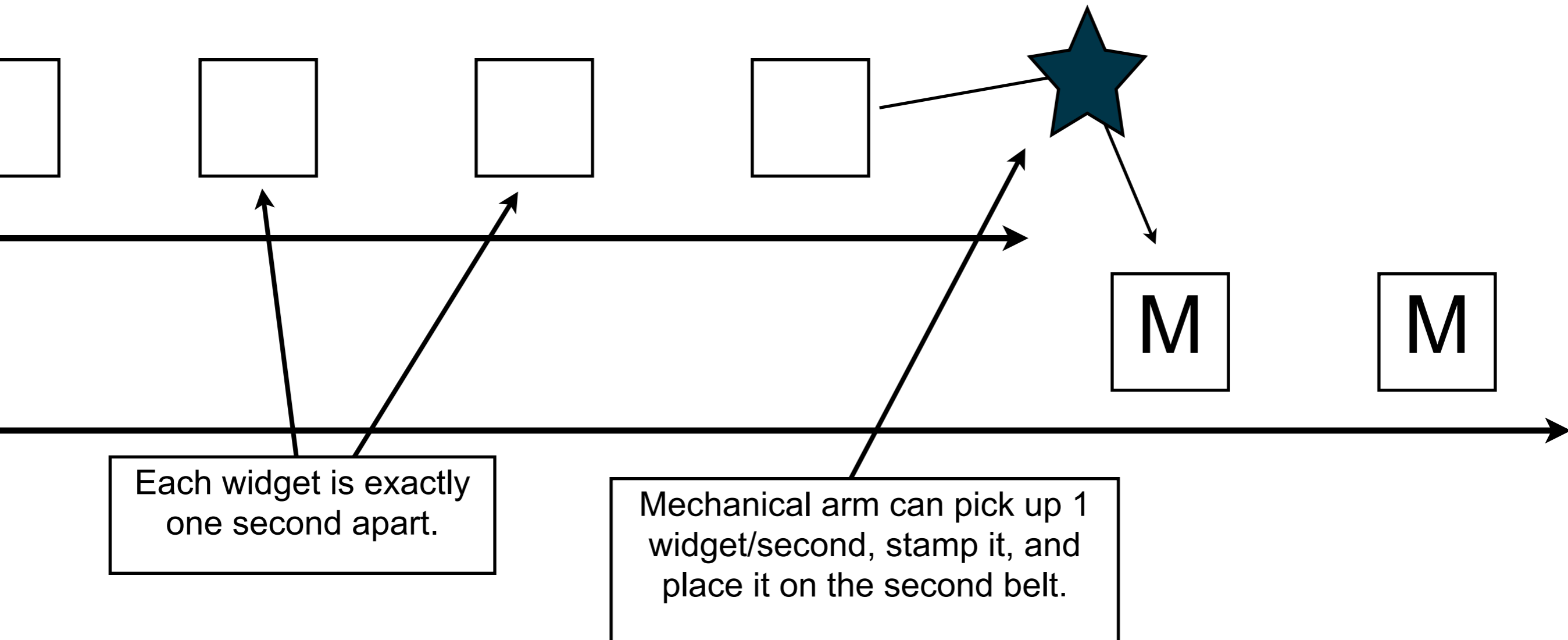
Next thing to know about CPUs

- ♦ Not all tasks arrive on time. Take the following example of a manufacturing process:



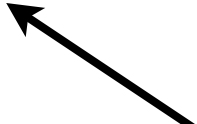
Next thing to know about CPUs

- ♦ Not all tasks arrive on time. Take the following example of a manufacturing process:



Throughput Question

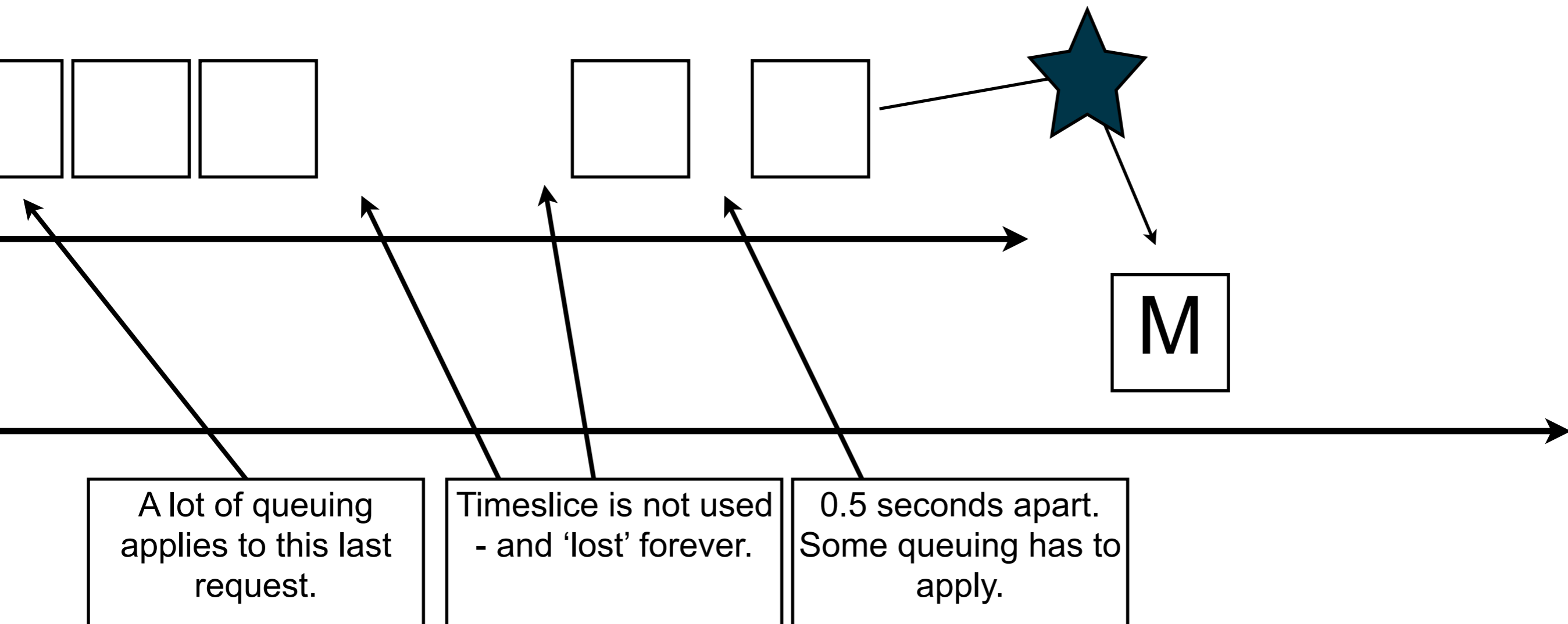
- ♦ There is only one mechanical arm - no parallelism is possible.
 - ♦ Service time of the mechanical arm is 1 second.
 - ♦ Maximum capacity is 60 boxes/minute.
- ♦ Can we have a throughput of 60 boxes/minute and a response time of 1 second?



In this example we can. But only because the arrival rate of the widgets is controlled.

Important Real-Life Difference

- ♦ The arrival rate of requests is not evenly distributed:



So there are some lessons

- ♦ If you have random arrivals - you may not be able to reach capacity **and** have an acceptable response time.
- ♦ All CPUs hitting 100% may never happen.
- ♦ Just because you don't see CPUs hitting 100% it does not mean that you do not have a problem.
 - ♦ There may still be a response time impact.

When Are CPU's A Problem?

- ◆ CPUs being used could be a good thing.
 - ◆ It shows the efficiency of a storage engine to be able to use resources - and not being blocked waiting.
- ◆ CPUs being close to maxed out could be the symptom of a very bad thing.
 - ◆ It could be that response time is suffering because the chance of queuing gets higher as utilization increases.
 - ◆ In many cases >75% utilization starts to have a very visible effect on response time.
- ◆ CPUs not being used could be good or bad
 - ◆ nobody is visiting your website
 - ◆ you've eliminated the database from your application

CPU

- ◆ Faster CPUs:
 - ◆ Recommended for databases:
 - ◆ In MySQL: 1 Statement is run by 1 thread == 1CPU
- ◆ More Cores provide more concurrency:
 - ◆ Watch out for MySQL CPU scalability limits (workload/MySQL version dependent)
 - ◆ Ballpark figure MySQL/InnoDB: 24 cores, used to be 4 a couple of years back (MySQL 5.0)
- ◆ Architecture:
 - ◆ Most Common: x86-64
 - ◆ 32bit is limited to 2.*GB of memory per process*
- ◆ Ok with Virtualization (except sharing resources)

CPU

- ◆ HyperThreading & Turbo-Boost
- ◆ Use VT when used in virtualized environments
- ◆ Disable Dynamic CPU Scaling, Set the clock speed to fixed

CPU Utilization

◆ vmstat

```
# vmstat 5
```

```
procs -----memory----- ----swap-- ----io----- -system-- ----cpu-----
r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa
9  7  2808 159360 378048 26494424  0  0  6  61  1  0  1  2 98  0
0  1  2808 166260 378140 26343512  0  0 22483 8739 6244 59414  9  4 61 26
0 11  2808 160064 378192 26188864  0  0 23728 4444 6191 71401 11  4 61 24
7 13  2808 164408 378236 26023756  0  0 24036 4618 6769 75098 11  4 57 29
7 10  2808 161044 378340 25860012  0  0 23203 8597 7266 84357 12  4 59 24
7  8  2808 167432 378404 25705900  0  0 20429 5858 7047 84135 13  4 61 22
7 12  2808 159216 378520 25565520  0  0 21101 11900 7494 89128 13  4 54 29
```

◆ mpstat

```
# mpstat -P ALL 5
```

```
10:36:12 PM CPU %user %nice %sys %iowait %irq %soft %steal %idle intr/s
10:36:17 PM all 18.81 0.05 3.22 0.22 0.24 2.71 0.00 74.75 13247.40
10:36:17 PM 0 19.57 0.00 3.52 0.98 0.20 2.74 0.00 72.99 1939.00
10:36:17 PM 1 18.27 0.00 3.08 0.38 0.19 2.50 0.00 75.58 1615.40
10:36:17 PM 2 19.09 0.20 3.35 0.20 0.39 1.97 0.00 74.80 1615.60
10:36:17 PM 3 17.73 0.00 3.47 0.39 0.39 3.08 0.00 74.95 1615.40
```

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ **Memory**
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL

Memory

- ◆ Why Does MySQL Need Memory?
- ◆ Where Does MySQL use Memory?
- ◆ Memory Fit
- ◆ Working Set
- ◆ Memory Or Disk Subsystem?

Why Does MySQL Need Memory?

- ◆ Per Session Buffering
 - ◆ sort buffer, temp tables
- ◆ Metadata/Locking/...
 - ◆ index statistics, table definitions
- ◆ Caching data
 - ◆ Decrease Reads: faster response time
 - ◆ Decrease Random Writes: queue writes in cache and do more sequential disk writes

Where Does MySQL Use Memory?

- ◆ Filesystem Cache
 - ◆ binary/relay log
 - ◆ **MyISAM data** is not buffered in MySQL, it relies on filesystem cache
- ◆ MySQL Cache:
 - ◆ InnoDB: **innodb_buffer_pool_size** (pages),
`innodb_log_buffer_size`
 - ◆ MyISAM: **key_buffer_size** (indexes only)
- ◆ Per Session Buffers:
 - ◆ `sort_buffer_size`, `join_buffer_size`,
`read_buffer_size`, `read_rnd_buffer_size`,
`tmp_table_size`

Where Does MySQL Use Memory?

- ◆ # free -m

	total	used	free	shared	buffers	cached
Mem:	32177	30446	1730	0	368	16649
-/+ buffers/cache:		13428	18748			
Swap:	4095	2	4093			

- ◆ # SHOW ENGINE INNODB STATUS\G

```
-----  
BUFFER POOL AND MEMORY  
-----  
Total memory allocated 4648979546; in additional pool allocated  
16773888  
Buffer pool size      262144  
Free buffers          0  
Database pages       258053  
Modified db pages    37491  
Pending reads        0  
Pending writes: LRU 0, flush list 0, single page 0  
Pages read 57973114, created 251137, written 10761167  
9.79 reads/s, 0.31 creates/s, 6.00 writes/s  
Buffer pool hit rate 999 / 1000
```

Memory Fit

- ◆ When all data fits in memory:
 - ◆ No reads have to come from disk
 - ◆ Remember response time of disks compared to memory?
- ◆ Does all your data need to fit in memory?

Working Set

- ♦ The size/percentage of the data that is frequently used
or
- ♦ How much of the data do we read over a certain period?

- ♦ Application dependent
- ♦ Between 1% and 100% of total data size

Working Set & Memory

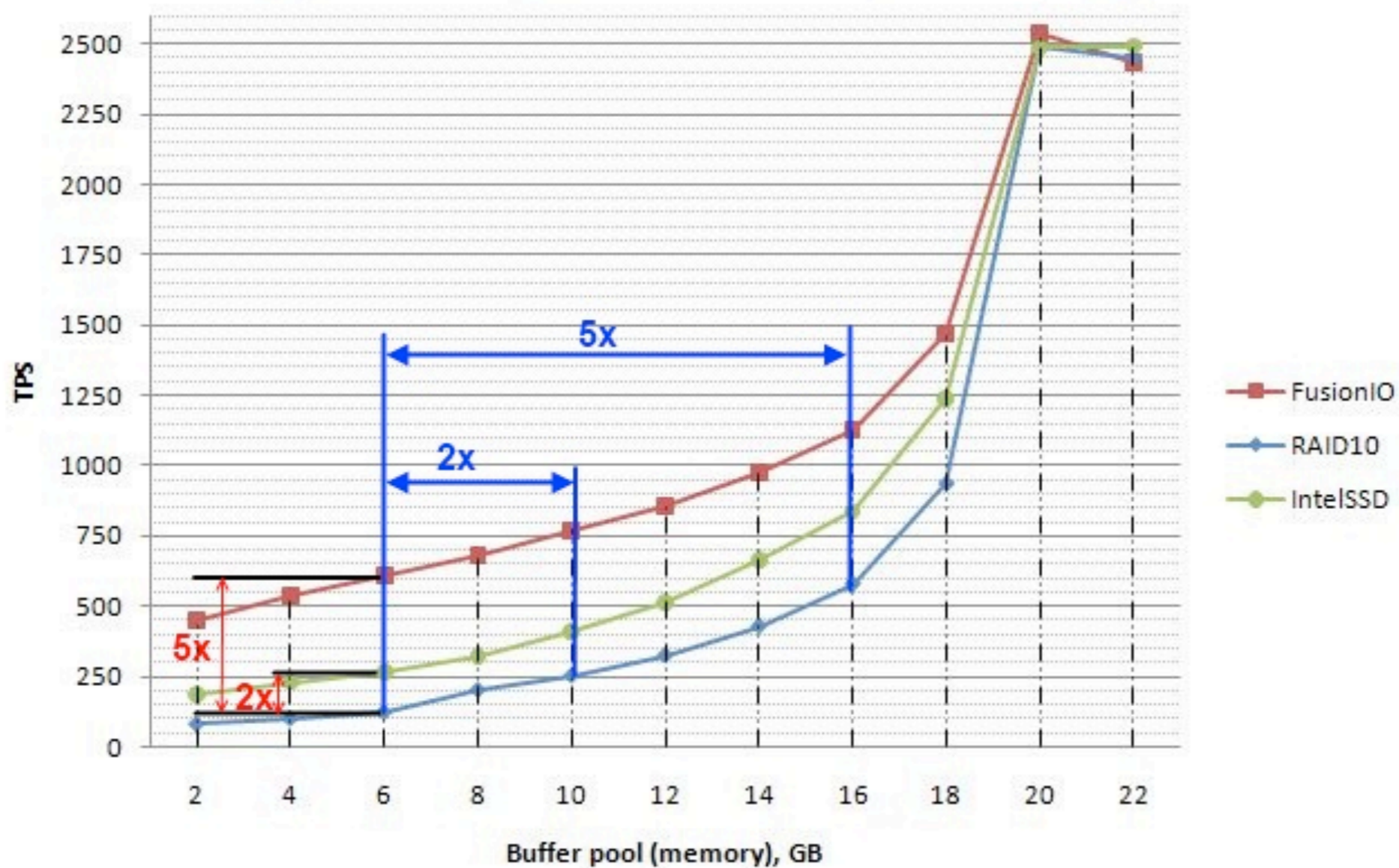
- ◆ When working sets fits in memory
 - ◆ hardly any reads have to come from disk
- ◆ When working set does not fit in memory:
 - ◆ Determine acceptable memory<->disk ratio
 - ◆ Dependent on response time/throughput requirements
 - ◆ Benchmark!
 - ◆ How much memory do we need to optimize? Is it cost-effective?

Decreasing Working Set

- ◆ Query Optimization
- ◆ Archive data
- ◆ Compression
- ◆ Correct data types?

Memory or Disk?

sysbench oltp, 80mln rows (18GB data)



<http://www.mysqlperformanceblog.com/2010/04/08/fast-ssd-or-more-memory/>

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ **Disk**
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL

Remember The Numbers

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Mental Math..

- ♦ $10,000,000 \text{ ns} = 10\text{ms} = 100 \text{ operations/second}$.
 - ♦ The figure quoted here is about the **average** for a 7200RPM drive.
 - ♦ When we talk about our storage devices, we most commonly measure them in IOPS (IO operations per second).
 - ♦ So a 7200RPM drive can do approximately 100IOPS.
 - ♦ 15K RPM disks might do ~160-180 IOPS.

Why count “operations”?

- ♦ Because there’s not much difference between doing one small request versus one slightly larger request. Again;

Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns

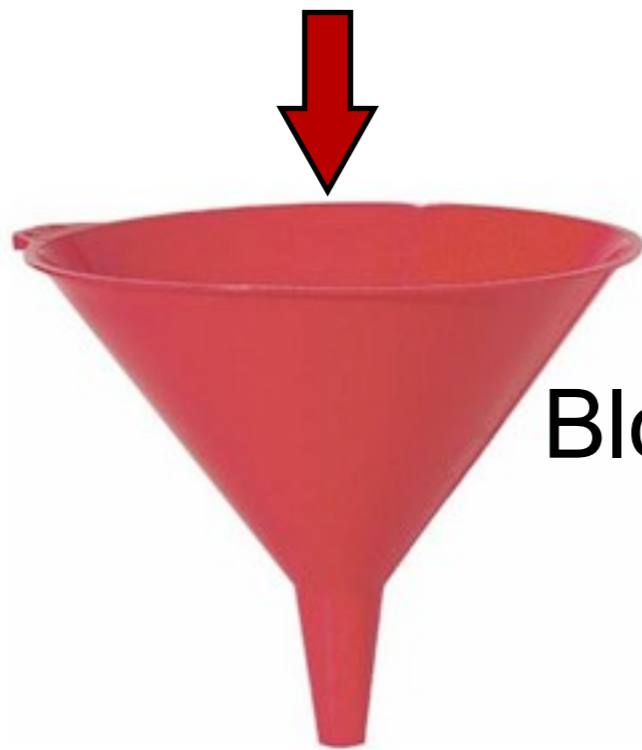
Yeah, there's a gap:

- ◆ **For each disk operation:**
 - ◆ Millions of CPU operations can be done.
 - ◆ Hundreds of thousands of memory operations can be done.

[os default] Everything is buffered!

- When you write to a file, here's what happens in the Operating System:

Block 9, 10, 1, 4, 200, 5.



Block 1, 4, 5, 9, 10, 200



What happens to this buffer if we lose power?

The OS provides a way!

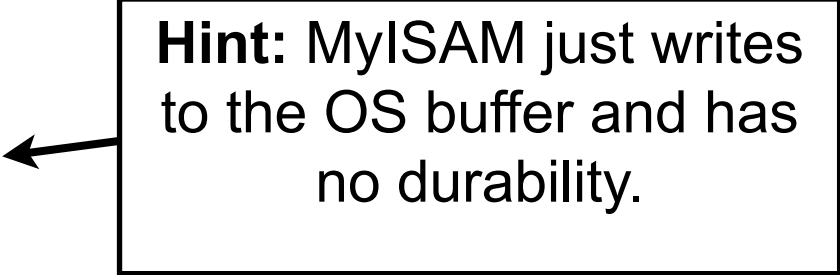
♦ `$ man fsync`

Synopsis

```
#include <unistd.h>
int fsync(int fd);
int fdatasync(int fd);
```

Description

`fsync()` transfers ("flushes") all modified in-core data of (i.e., modified buffer cache pages for) the file referred to by the file descriptor `fd` to the disk device (or other permanent storage device) where that file resides. The call blocks until the device reports that the transfer has completed. It also flushes metadata information associated with the file (see `stat(2)`).



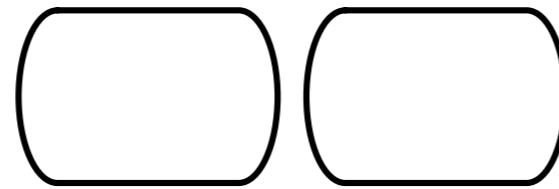
Hint: MyISAM just writes to the OS buffer and has no durability.

fsync and Virtualization

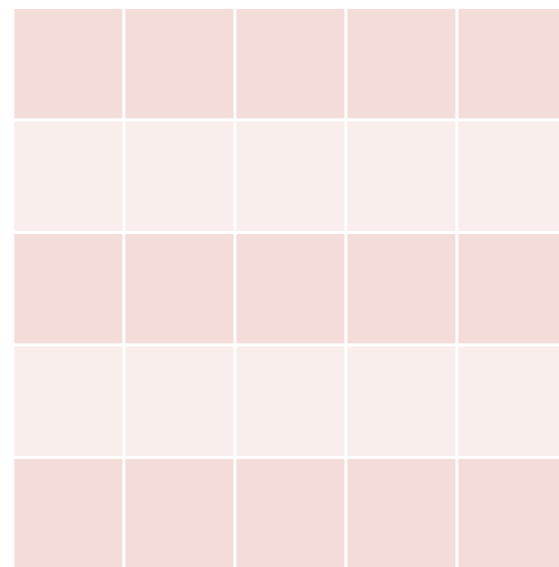
- ♦ Make Sure the hypervisor does not lie when a virtual machine does an fsync
 - ♦ On Linux hosts, VMware does not use unbuffered IO, because it is not safe or supported across all the Linux versions that VMware supports. So currently, VMware hosted products on Linux hosts always use buffered IO.
http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1008542

InnoDB Provides a way!

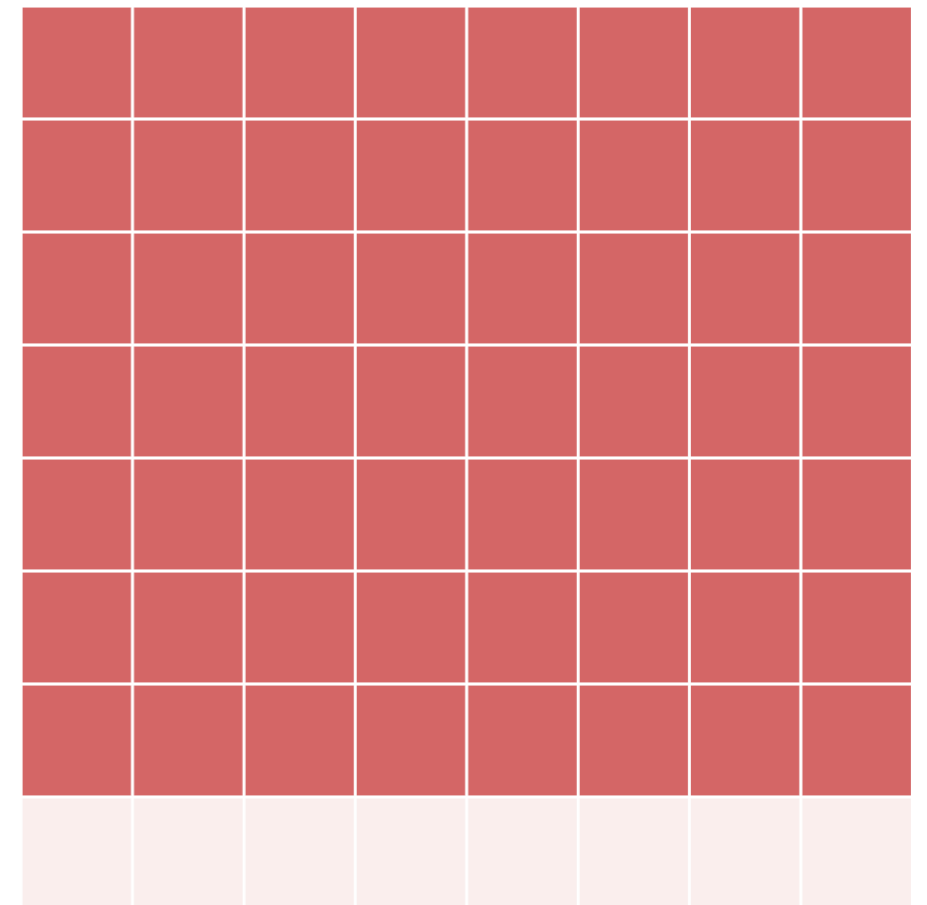
SELECT * FROM
Users WHERE
name='Fred'



Log Files

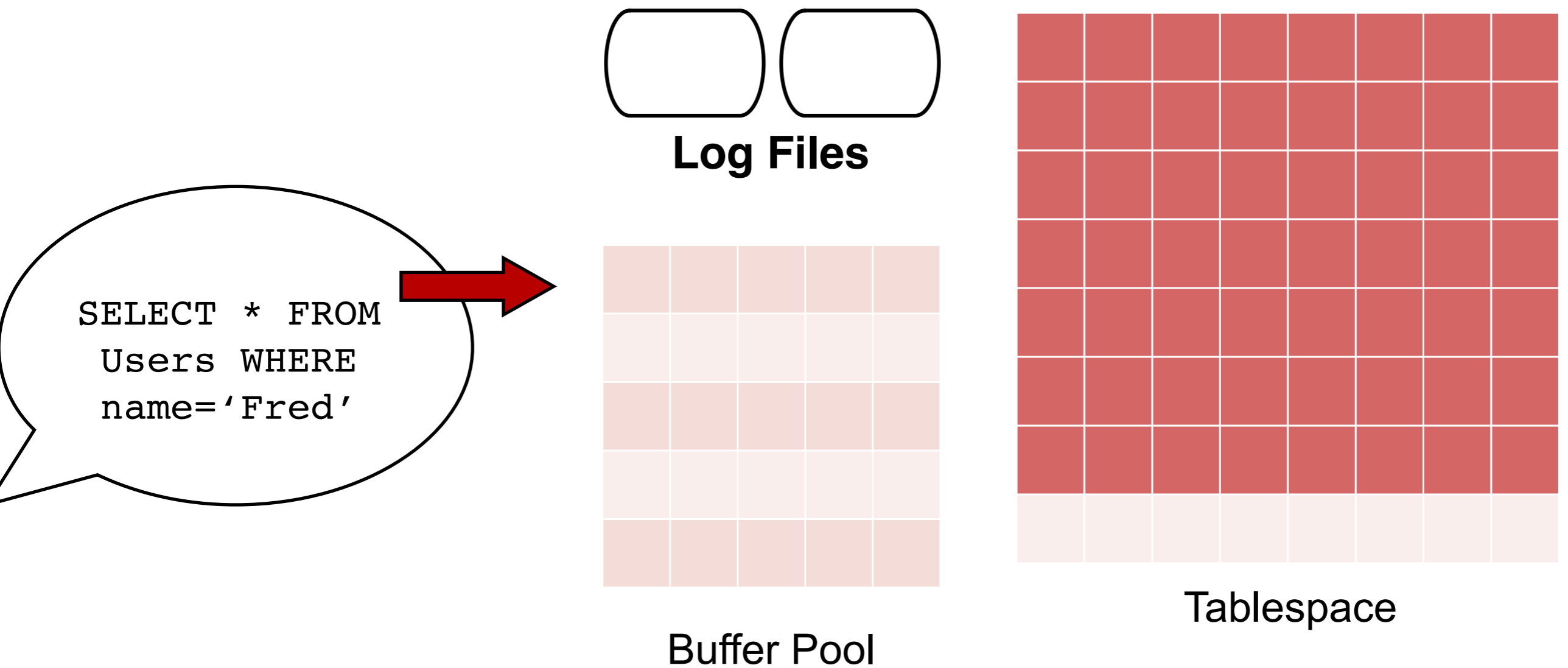


Buffer Pool

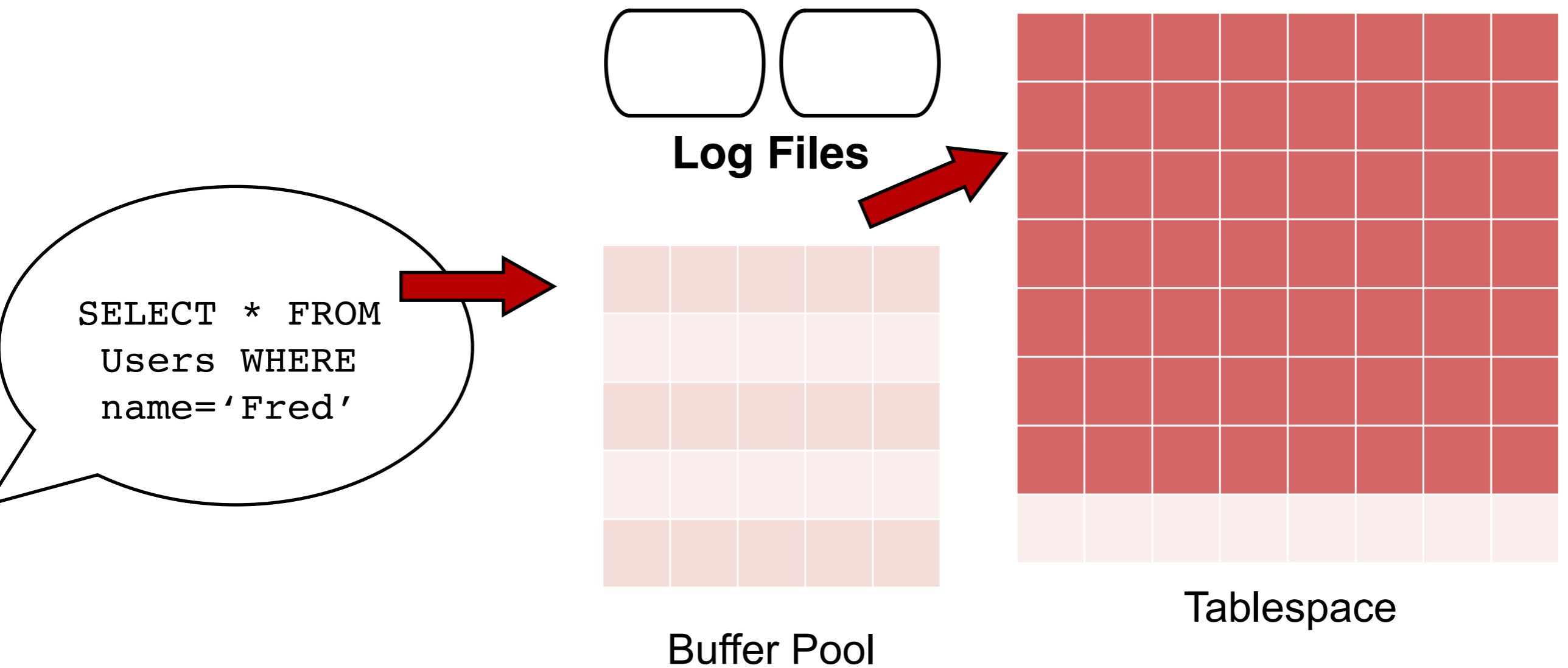


Tablespace

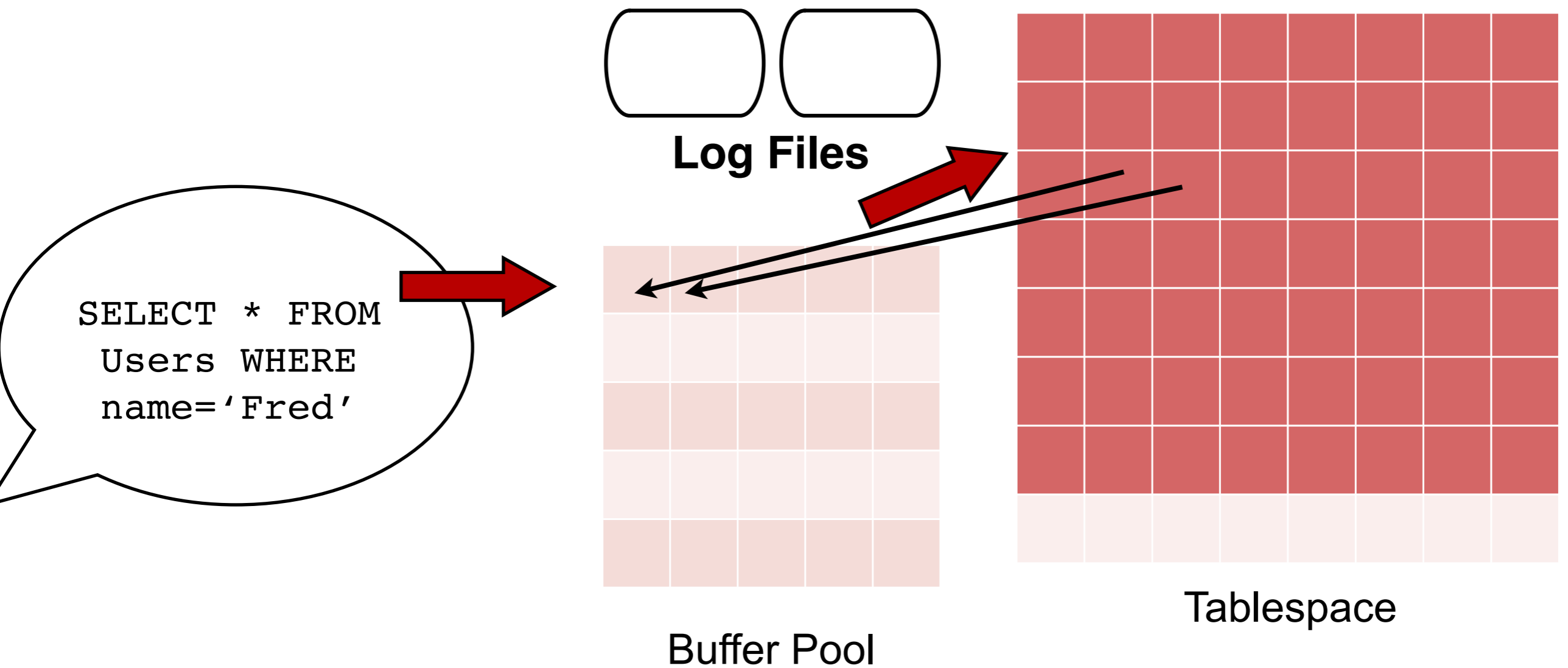
InnoDB Provides a way!



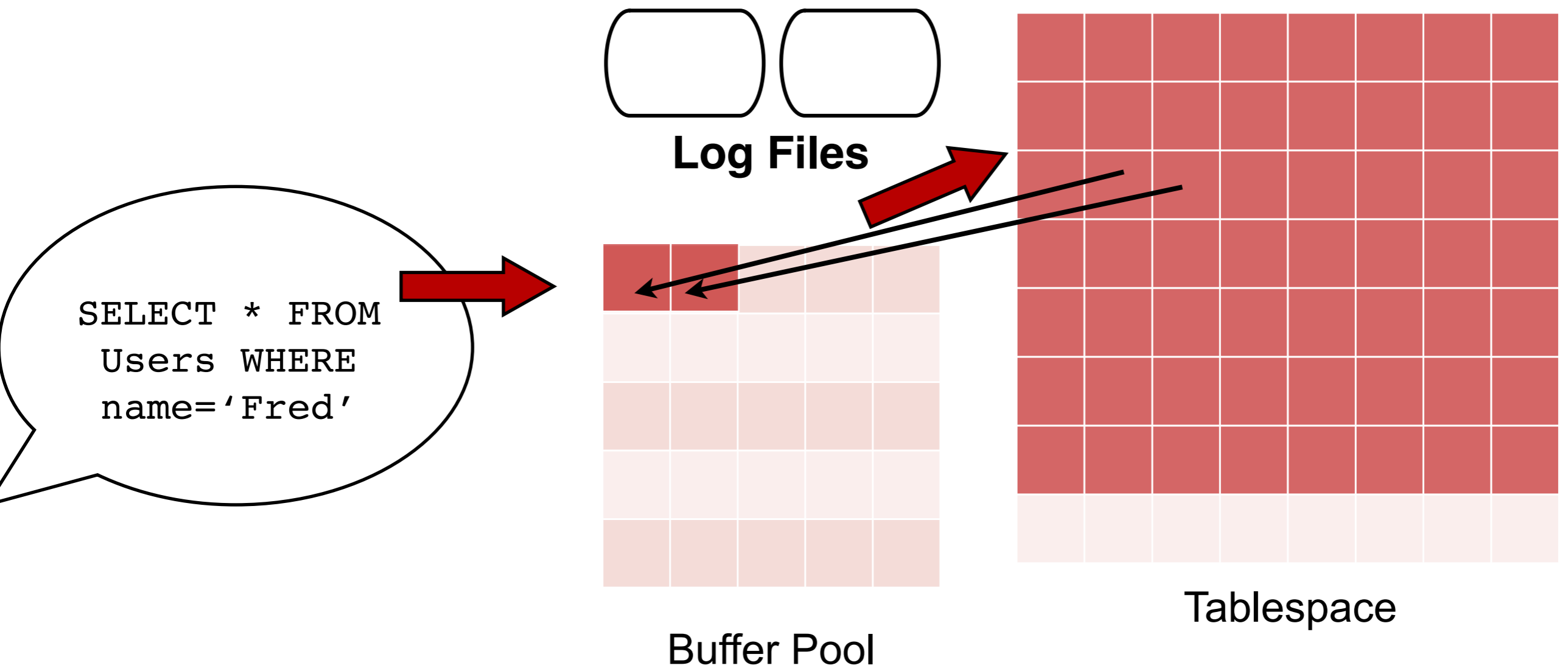
InnoDB Provides a way!



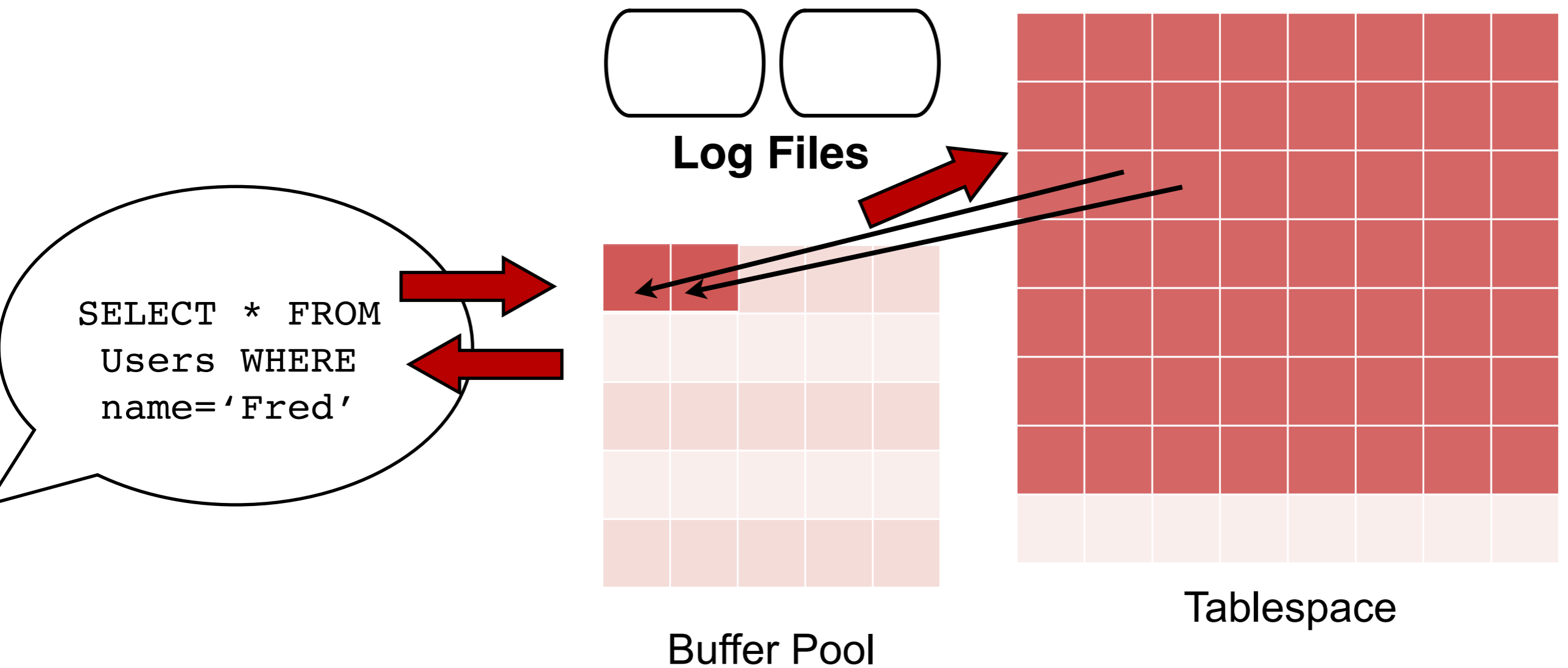
InnoDB Provides a way!



InnoDB Provides a way!

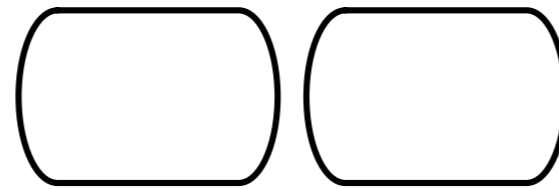


InnoDB Provides a way!

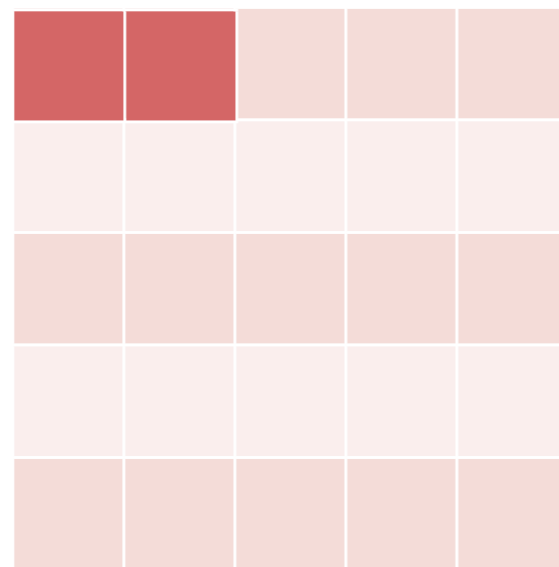


InnoDB (cont.)

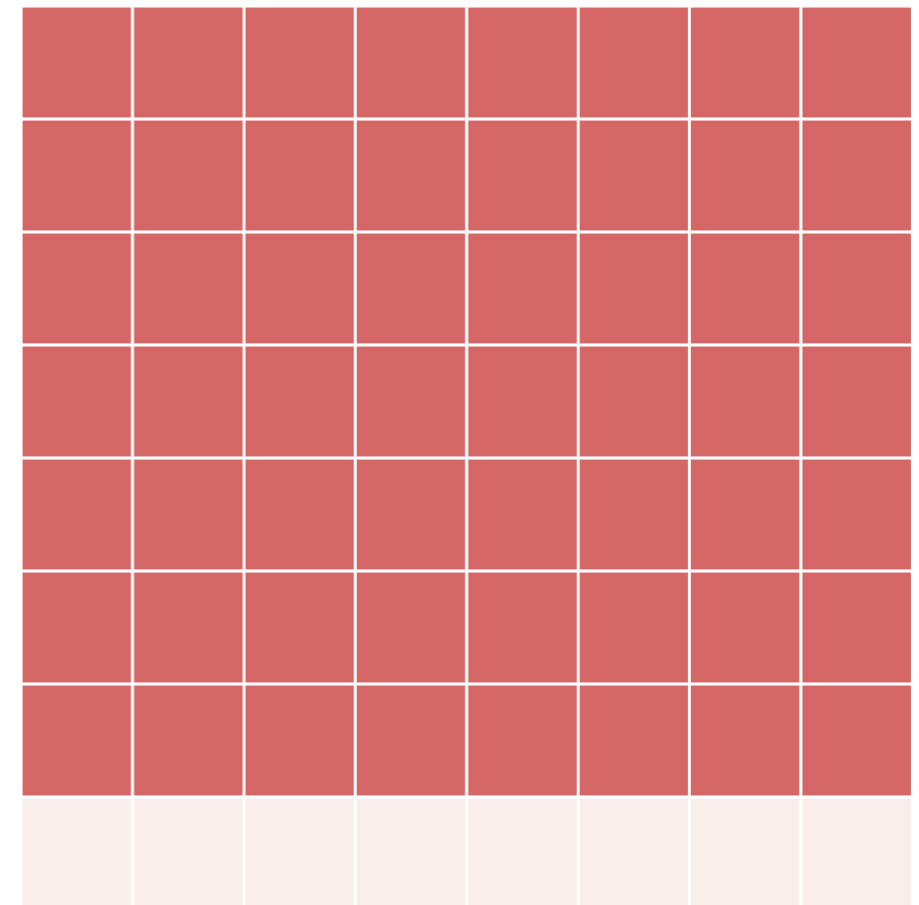
UPDATE User SET
name = 'leFred'
WHERE name =
'Fred'



Log Files

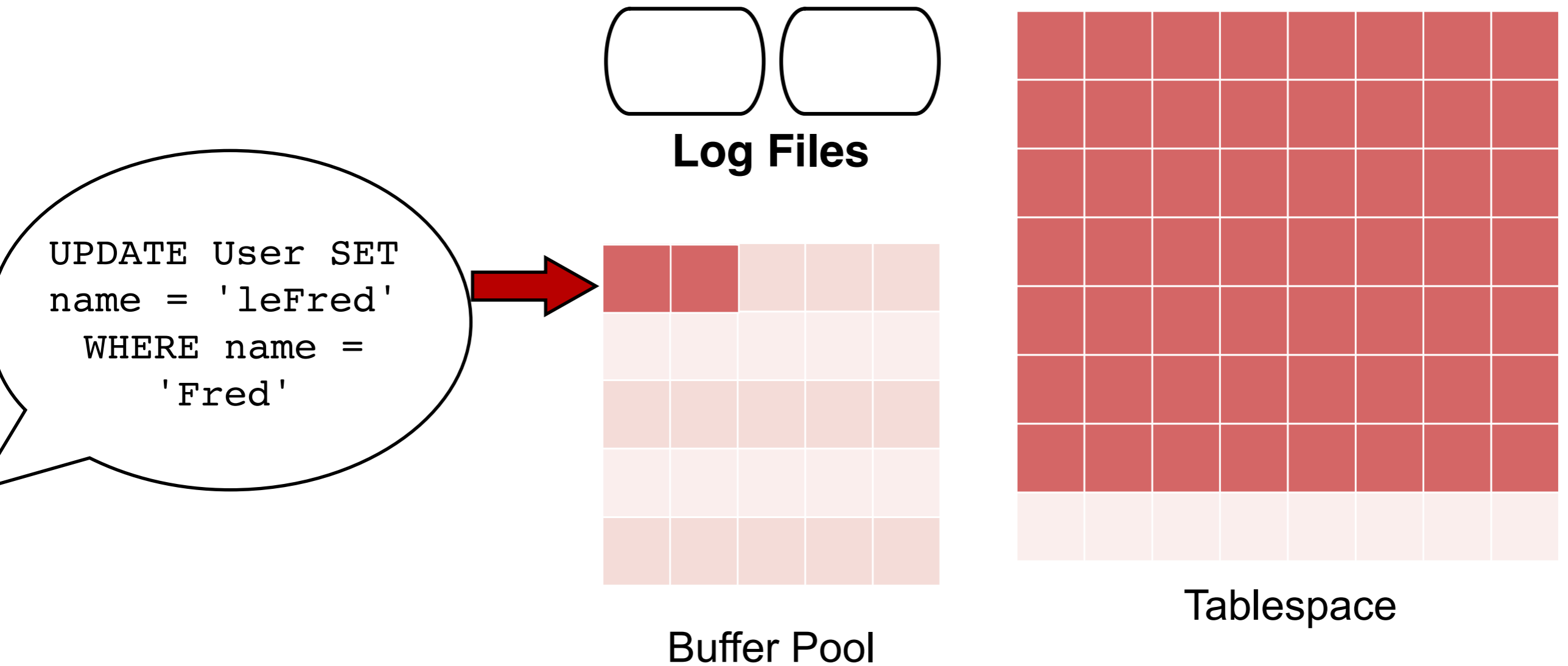


Buffer Pool

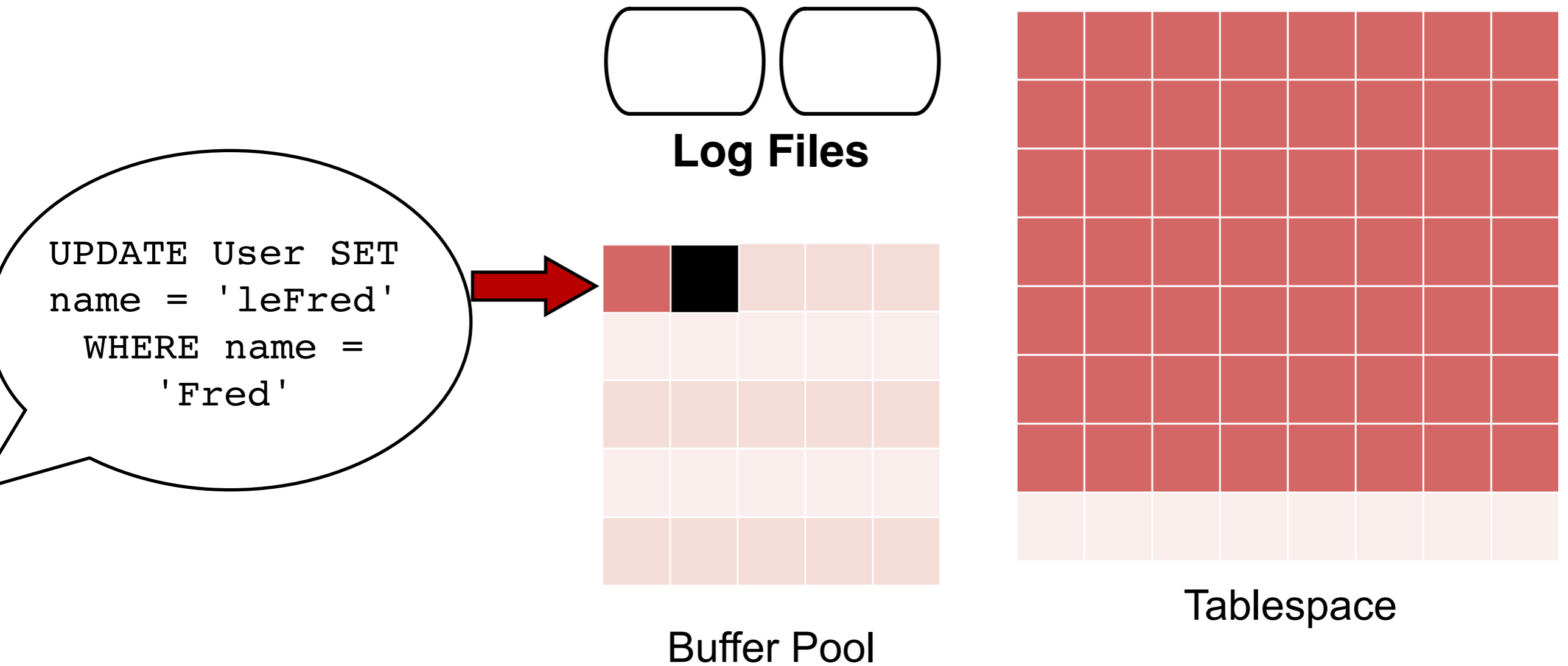


Tablespace

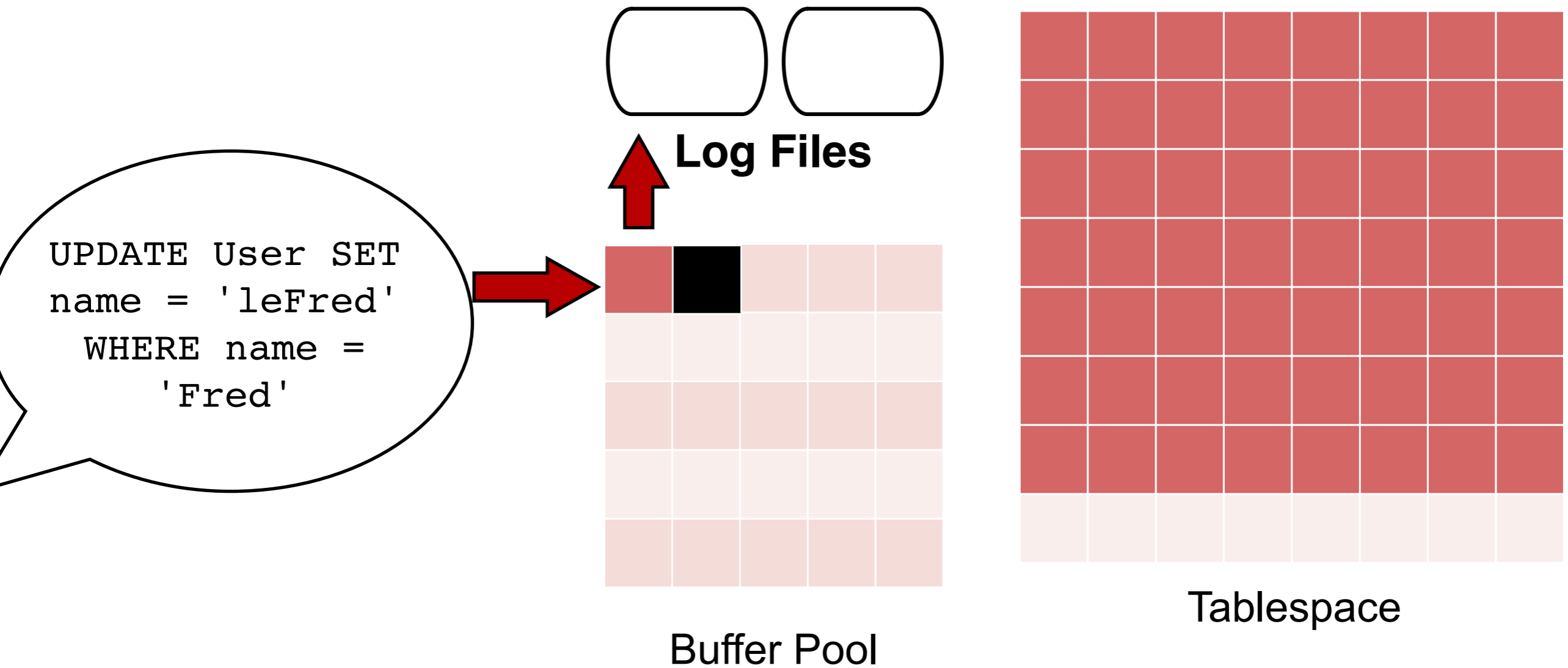
InnoDB (cont.)



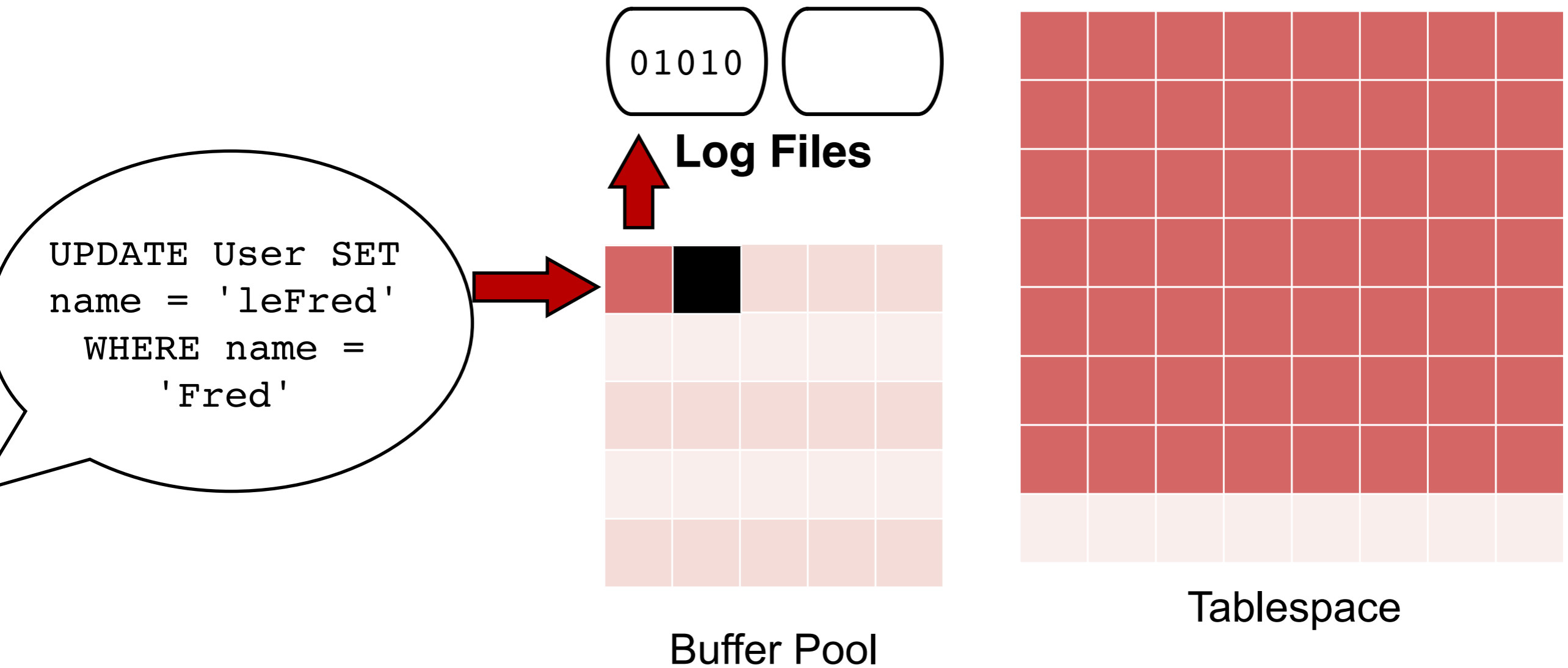
InnoDB (cont.)



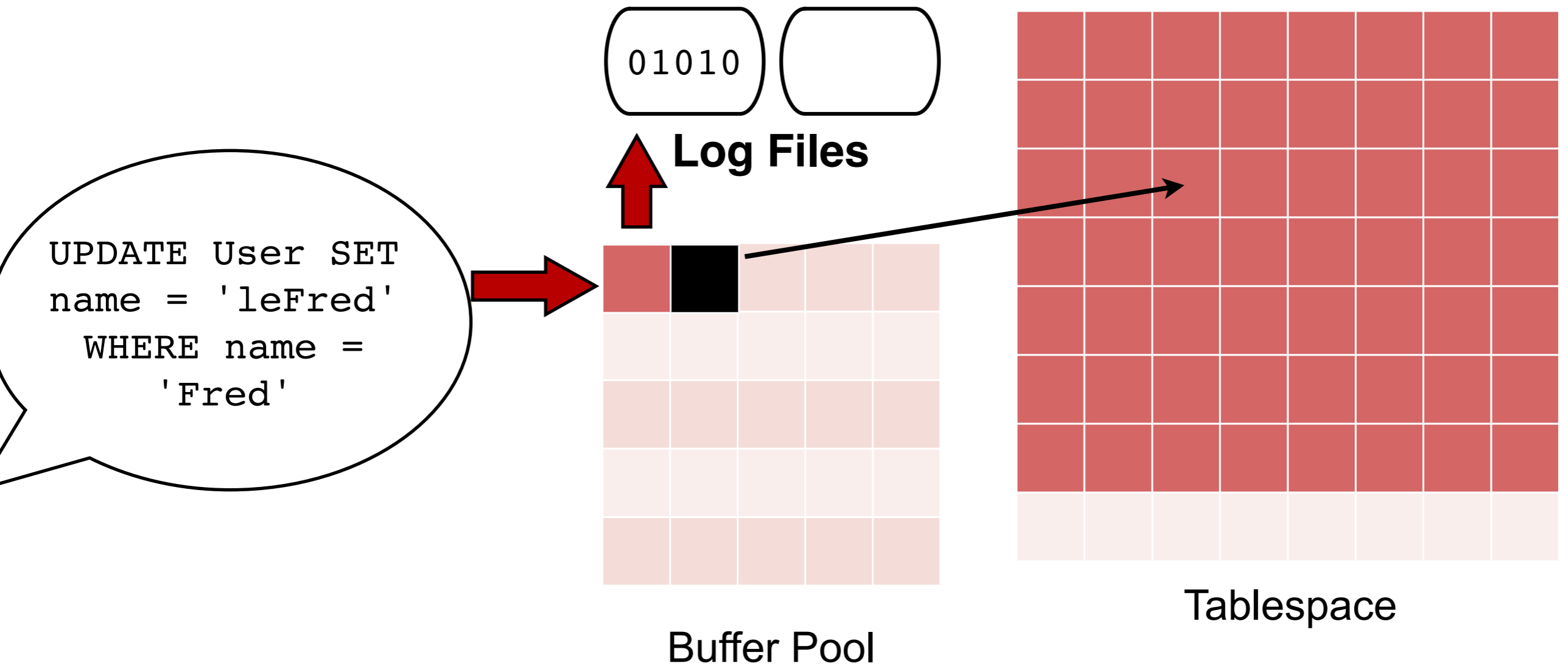
InnoDB (cont.)



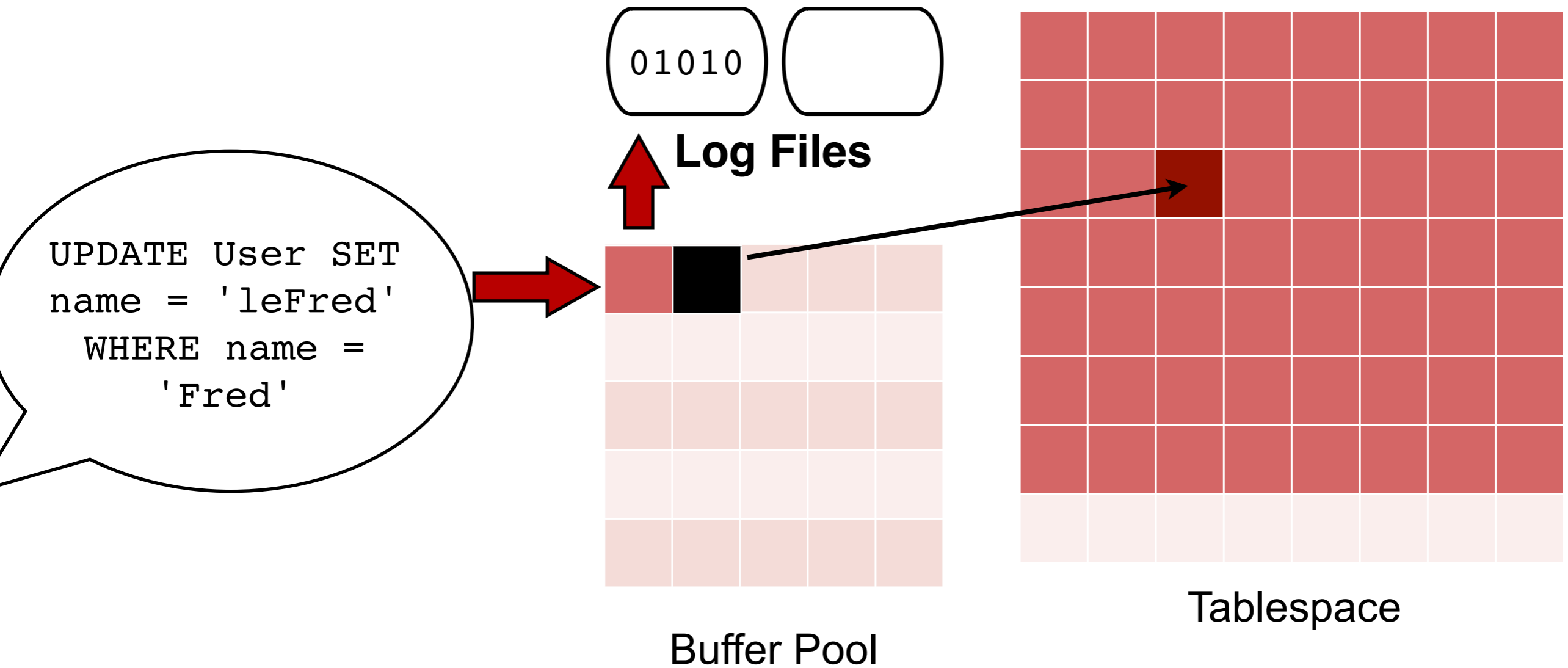
InnoDB (cont.)



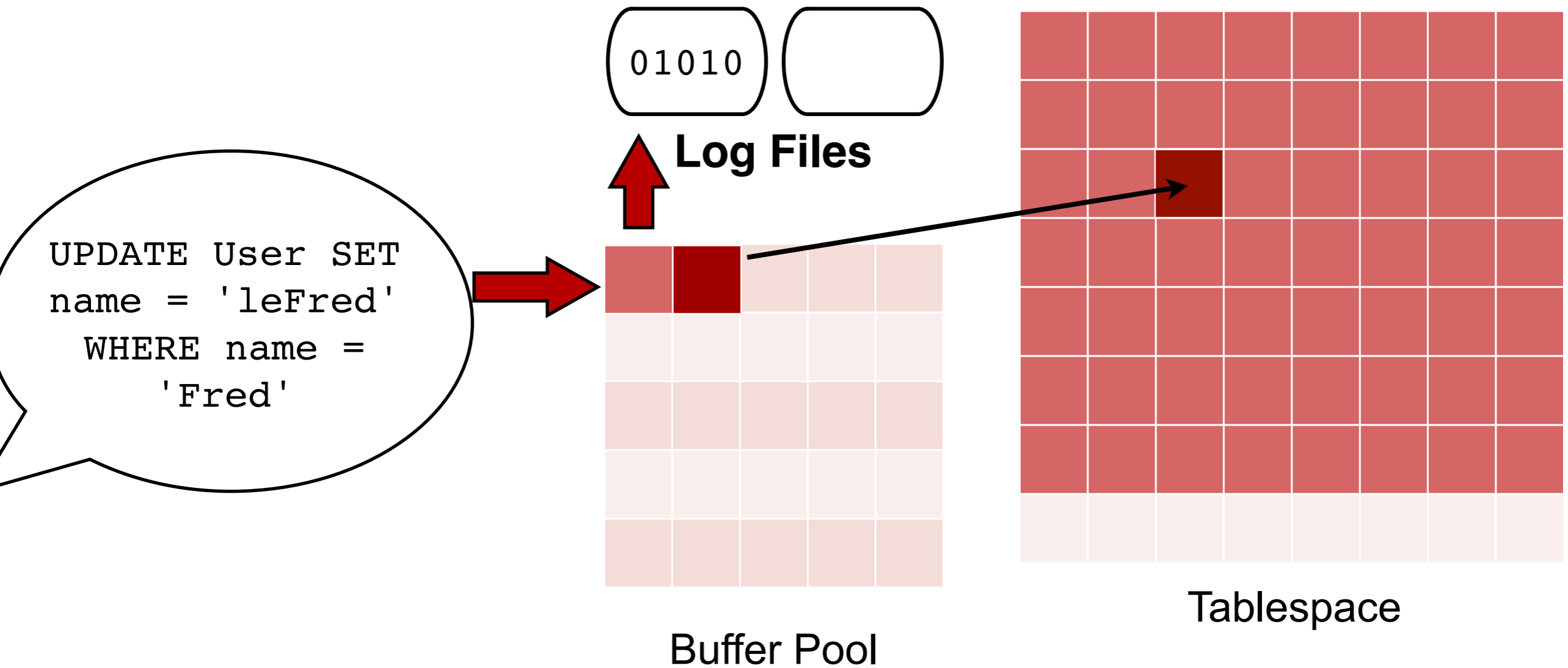
InnoDB (cont.)



InnoDB (cont.)



InnoDB (cont.)



Why does InnoDB work like...?

- ◆ Log files are sequential IO.
- ◆ Writing down “dirty pages” is reordered to be sequential IO instead of order of the operation.
- ◆ Many database systems actually work this way.

So now we've established...

- ◆ **That:**
 - ◆ Disks are slow.
 - ◆ Algorithms are designed to avoid them wherever possible.
- ◆ **The next question becomes:**
 - ◆ When is touching disks unavoidable?

First with reads

- ◆ Reads are able to use caches. Add more memory to improve hit efficiency.
- ◆ Remember Working Set!

Then with writes

- ◆ Writes are not able to use caches in the same way.
 - ◆ They can buffer changes (dirty pages) in memory, but the transaction log must be flushed to disk to guarantee durability.
 - ◆ **ACID:** InnoDB will flush the log buffer and wait on each commit. The syncing is often a performance bottleneck on systems without a RAID controller/write-back cache.

Linux Commands

♦ # ./pt-diskstats -d sdb1

```
#ts dev rd_mb_s rd_cnc rd_rt wr_mb_s wr_cnc wr_rt busy in_prg
{1} sdb1 0.3 1.0 14.9 1.6 2.2 7.2 96% 0
{1} sdb1 0.4 1.3 13.9 6.3 2.5 7.4 95% 0
{1} sdb1 0.5 0.9 6.8 2.7 4.9 11.9 96% 0
```

c

Enter a column pattern: wr.*

```
#ts device wr_s wr_avkb wr_mb_s wr_mrg wr_cnc
wr_rt
{1} sdb1 208.5 21.5 2.2 63% 1.1 5.4
{1} sdb1 208.5 21.5 2.2 63% 1.1 5.4
{1} sdb1 216.2 25.9 2.7 69% 2.0 9.2
```

c

Enter a column pattern: rd.*

```
#ts device rd_s rd_avkb rd_mb_s rd_mrg rd_cnc rd_rt
{1} sdb1 28.5 16.3 0.2 0% 1.6 55.0
{1} sdb1 104.8 9.4 0.5 0% 1.6 14.8
{1} sdb1 85.0 11.0 0.5 0% 1.5 17.6
{1} sdb1 105.7 10.3 0.5 0% 2.1 19.7
```

<http://www.percona.com/doc/percona-toolkit/2.0/pt-diskstats.html>

SSD & Flash

◆ **SSD:**

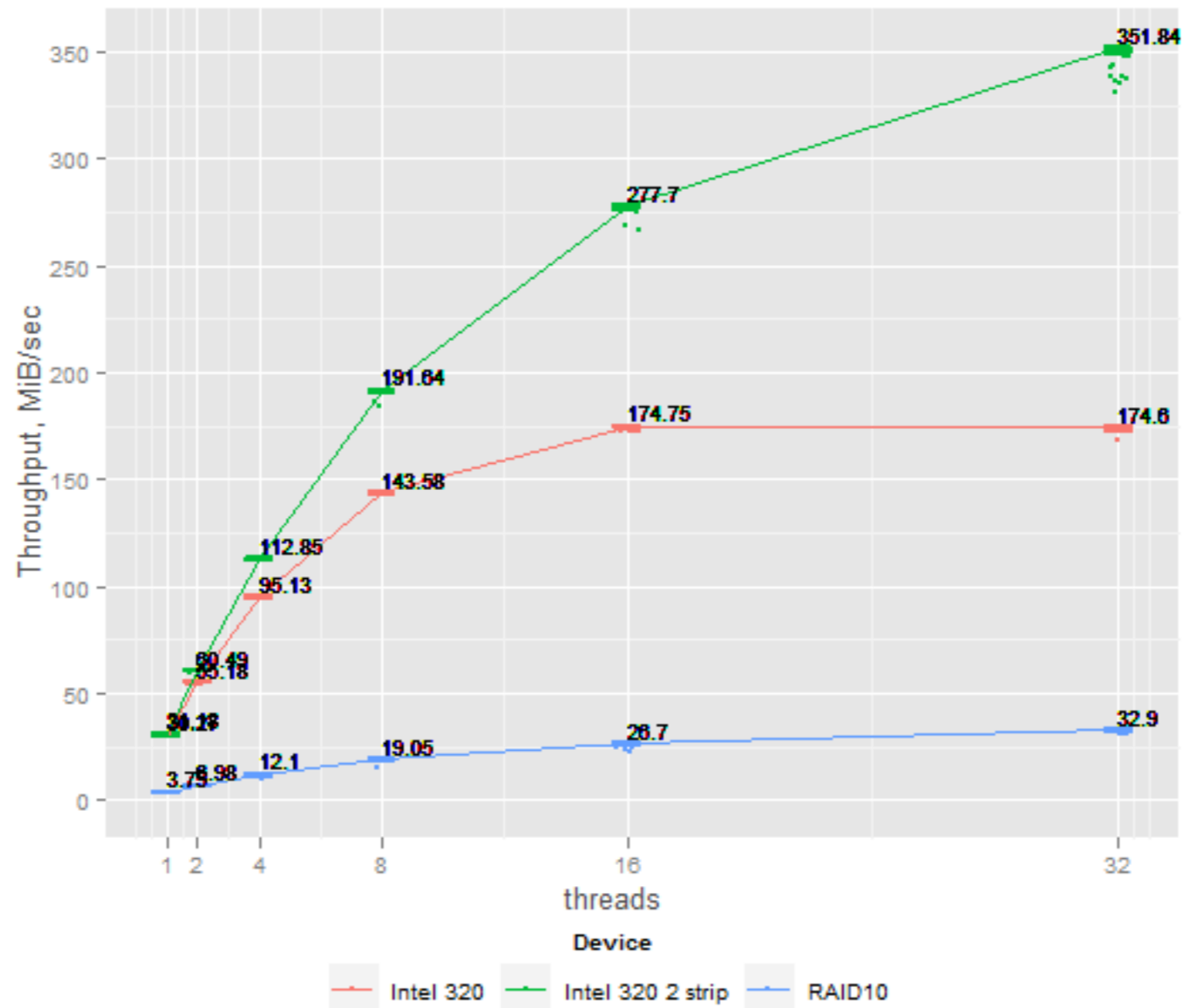
- ◆ Much More IOPS than traditional disks (x.000IOPS)
- ◆ Lower latency (~0.1ms)
- ◆ Hardware RAID and SSDs
 - ◆ Old HW RAID controllers are optimized to reduce random IO and do not expect disks that support x.000 IOPS

◆ **Flash:**

- ◆ Even more IOPS than SSDs (x0.000IOPS)
- ◆ Lower latency (~0.05ms)

SSD & Flash

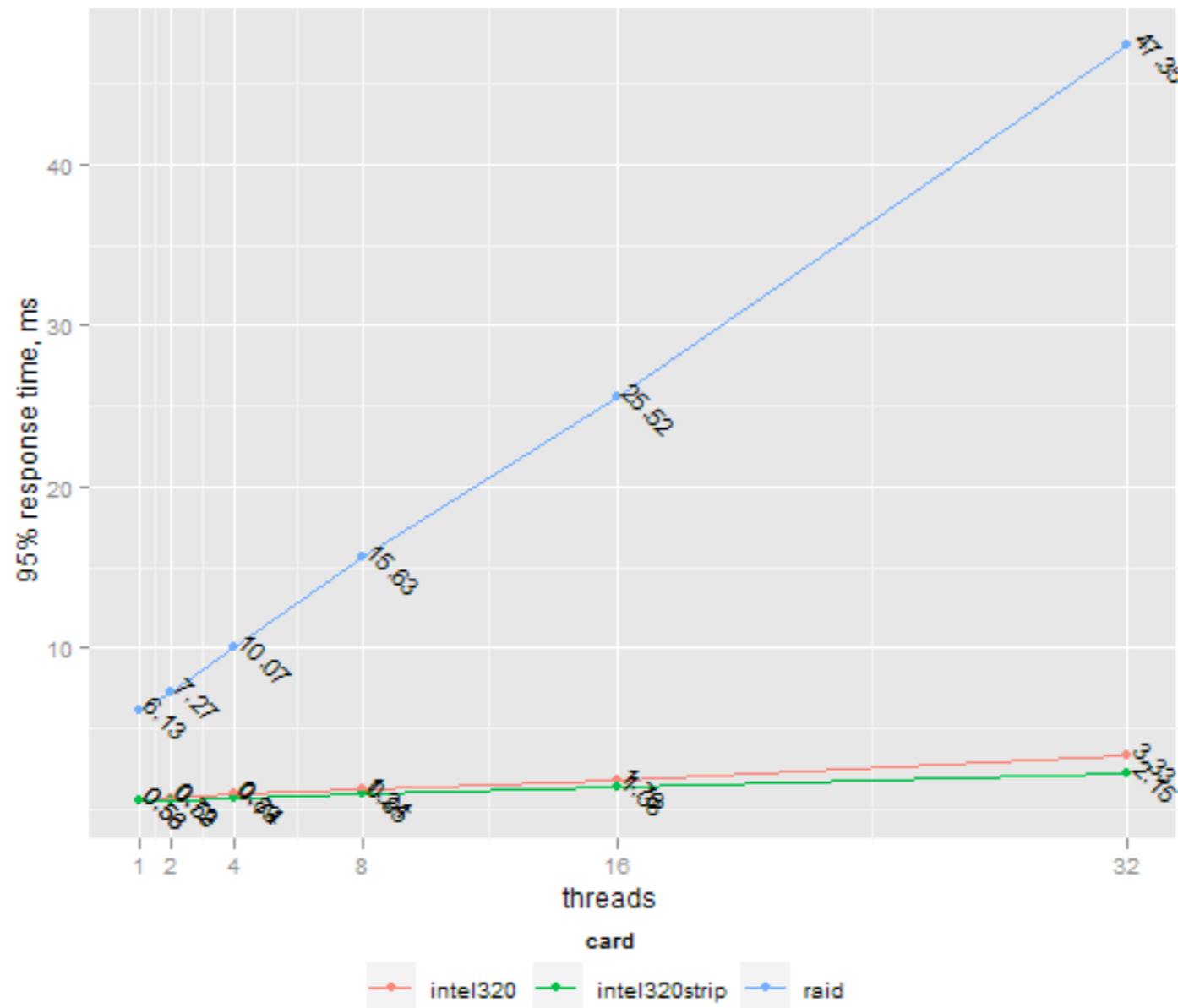
- ◆ Random Read Benchmark: Throughput:



<http://www.ssdperformanceblog.com/2011/07/intel-320-ssd-read-performance/>

SSD & Flash

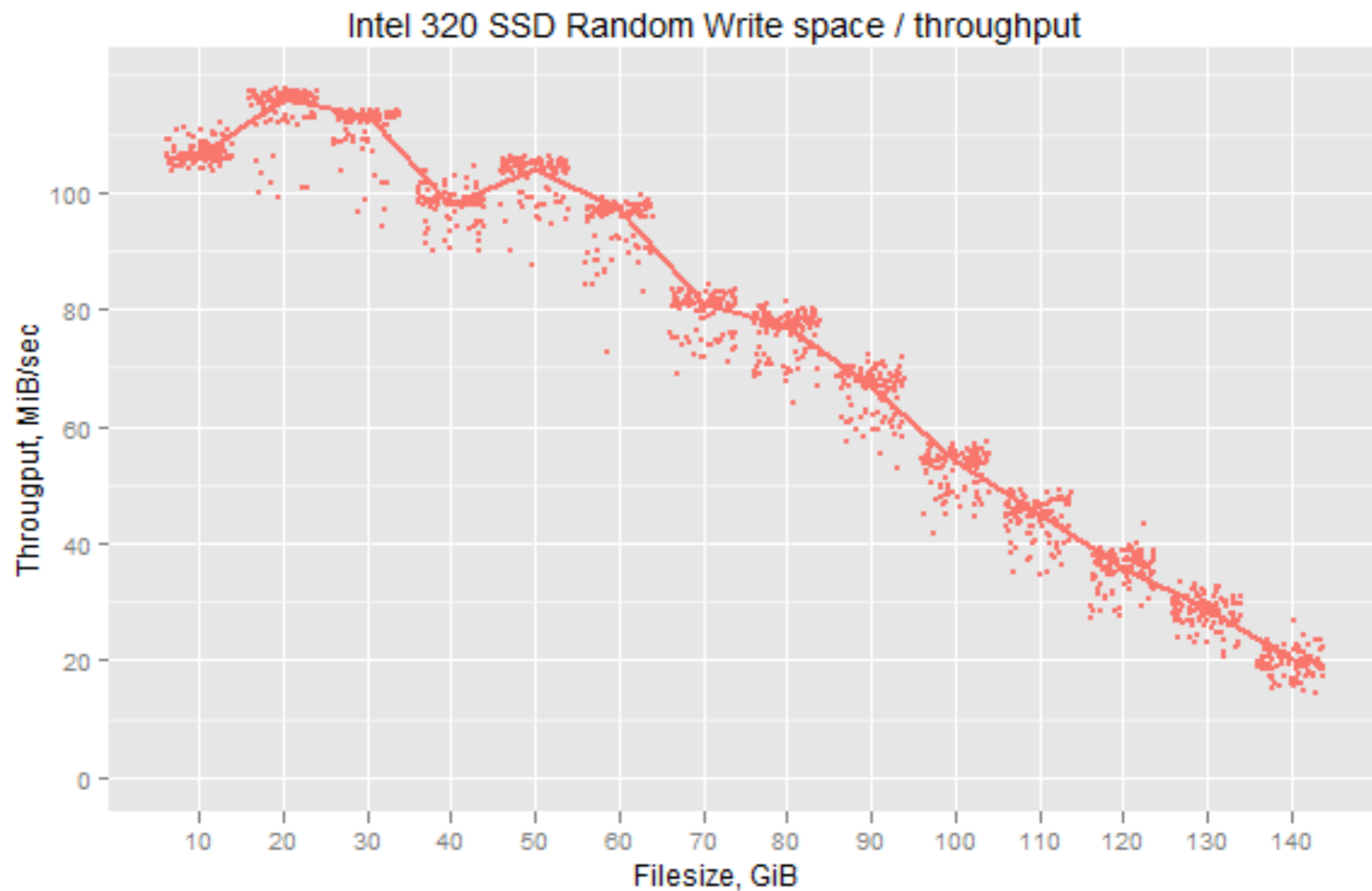
- ♦ Random Read Benchmark: Response Time:



<http://www.ssdperformanceblog.com/2011/07/intel-320-ssd-read-performance/>

SSD & Flash

- ◆ Performance decreases when they get more full:



When to use Flash or SSD

- ♦ When you cannot afford high latency cache misses
- ♦ Impossible to remove cache misses (too big working set)
- ♦ Server restart means empty caches: warming caches can take a long time, flash/ssds help (Percona Server also support storing/loading buffer pool contents: http://www.percona.com/doc/percona-server/5.5/management/innodb_lru_dump_restore.html)

RAID Levels

◆ Comparison:

Level	Faster Writes	Faster Reads	Other
RAID0	Yes	Yes	No redundancy!!
RAID1	No	Yes	
RAID5	* 1 write: 2 writes, 2 reads * Slower for random writes * Better for sequential writes	Yes	Slow recovery, \$\$-aware
RAID10	Yes	Yes	
RAID 50	Yes	Yes	Combination of 5 and 10

Disk Cache

- ◆ Write-Through:
 - ◆ Not recommended
 - ◆ Let MySQL/InnoDB handle the caching
 - ◆ Decreases cache left for write-back
- ◆ Write-Back
 - ◆ Necessity! fsyncs can be performed using the cache (~0.1ms) and not use IOPS from the disk
 - ◆ Requires Battery Backup Unit to be Durable
(beware of auto-learning:<http://www.mysqlperformanceblog.com/2010/11/16/on-the-perils-of-uniform-hardware-and-raid-auto-learn-cycles/>)
- ◆ Disk Write-Back Cache:
 - ◆ disable when the disk itself does not have capacitor or battery

SAN & NAS

- ◆ SAN:
 - ◆ Enterprisey
 - ◆ Higher Cost
 - ◆ More throughput because more disks
(why not DAS?)
 - ◆ Higher Latency
- ◆ NAS: Network mount
 - ◆ Latency
 - ◆ Locking/Syncing: reliable? Test
- ◆ Watch out when sharing resources!

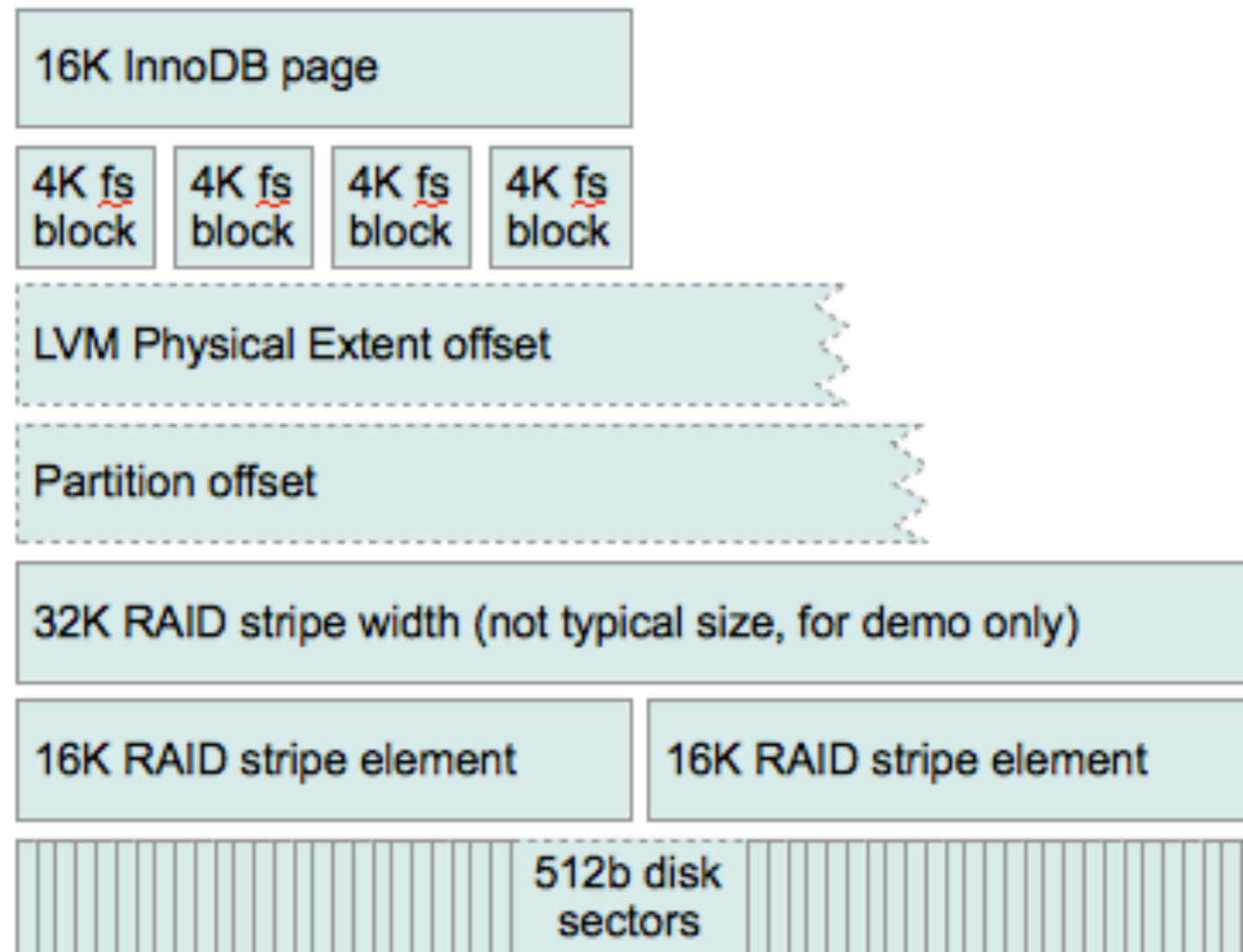
Combining Different Types

- ◆ InnoDB Transaction Logs, log files, binary log, InnoDB doublewrite* are written sequentially
- ◆ If Flash/SSD or Many Disks (20?):
Might be good to put on separate RAID1 with HDD (who are fast with sequential writes)

* Percona Server XtraDB Only

Aligning IO

Storage stack:



<http://www.mysqlperformanceblog.com/2011/06/09/aligning-io-on-a-hard-disk-raid-the-theory/>
<http://www.mysqlperformanceblog.com/2011/06/09/aligning-io-on-a-hard-disk-raid-the-benchmarks/>

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ **Network**
- ◆ Amazon Cloud
- ◆ Running MySQL

Network

- ◆ Use 1Gigabit or more:
 - ◆ More **Bandwidth**/Throughput
 - ◆ Lower **Latency**
- ◆ Trunking/Bonding:
 - ◆ High Availability
 - ◆ More Throughput

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ **Amazon Cloud**
- ◆ Running MySQL

Amazon EC2 EBS&RDS

- ◆ EC2:
 - ◆ Limited in types of instances: up to 68GB memory + eight-core 2 x Intel Xeon (cc2.8xlarge)
- ◆ EBS:
 - ◆ Unpredictable IO performance
 - ◆ Use RAID10 or RAID0 (does amazon mirror?)
- ◆ RDS:
 - ◆ Similar performance between EBS RAID10 MySQL 5.1 and RDS, Percona Server performs a bit better than RDS
 - ◆ Amazon manages your MySQL
 - ◆ Less flexibility (no login, no choice in choosing distro/version, less troubleshooting options)

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ **Running MySQL**

Running MySQL

- ◆ Usual Suspects
- ◆ Slaves
- ◆ Multiple MySQL Instances
- ◆ Hardware Is Not The Only Solution

The Usual Suspects

- ◆ Disk IO
 - ◆ When data size is too large (working set) compared to available caches
 - ◆ Bad Queries
- ◆ CPU
 - ◆ CPU-intensive operations
 - ◆ Resource Intensive Queries
- ◆ Does not mean adding more IO capacity or CPU's will resolve everything!

Slaves

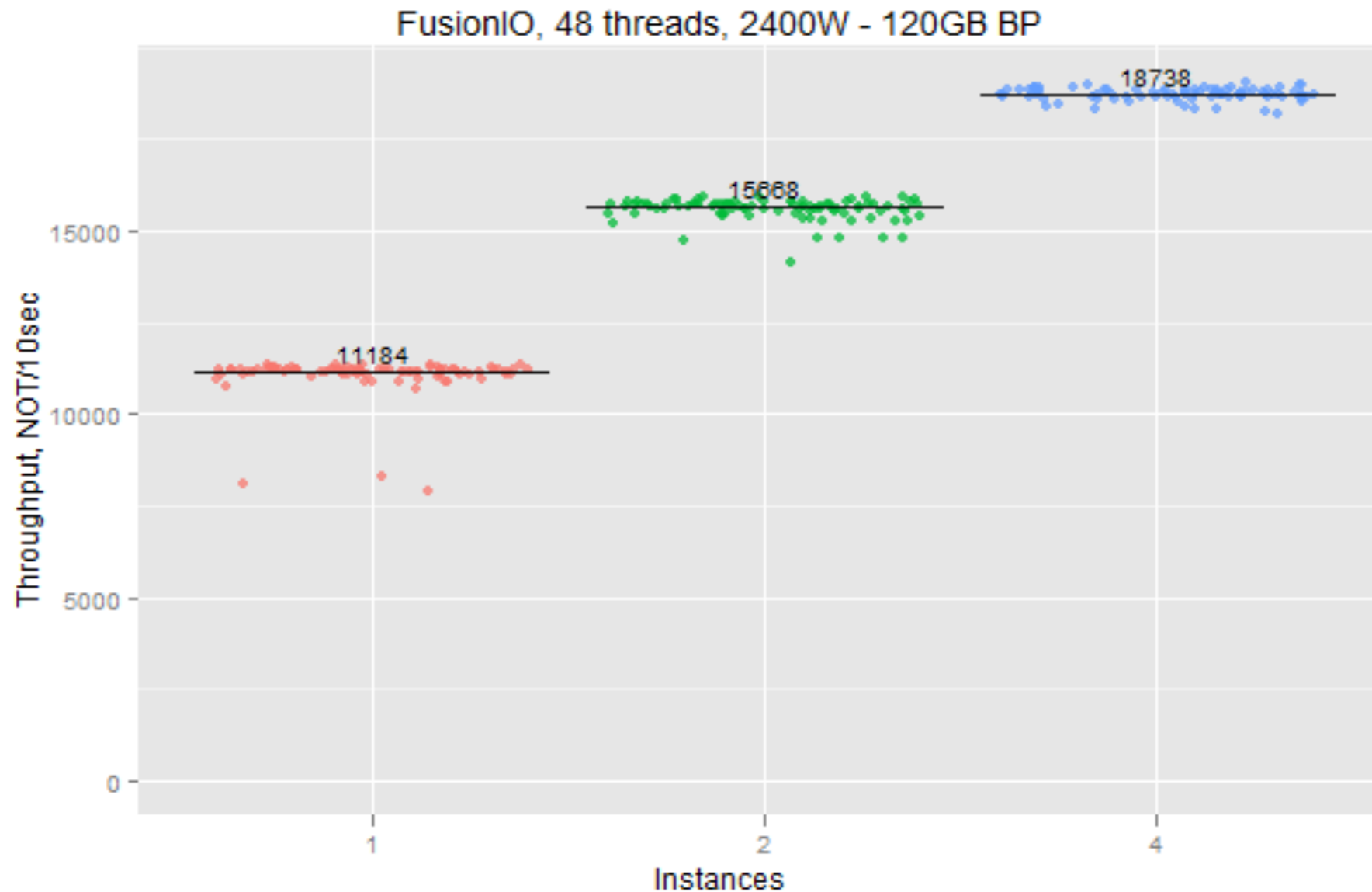
- ♦ Replication is single threaded: fast cpu's, faster disk (lower response time) help
- ♦ Might need less or more memory: working set could be smaller or bigger, depending on use

Multiple MySQL Instances

- ◆ When having 'high-end' machines with:
 - ◆ fast disks
 - ◆ many many cores
- ◆ PRO:
 - ◆ Better use of available resources
- ◆ CON:
 - ◆ 'sharing resources'
 - ◆ Operational Complexity
 - ◆ Splitting up dataset: sh..ar..ding

Multiple MySQL Instances

- ◆ Might Help. Benchmark!



<http://www.ssdperformanceblog.com/2011/10/multiple-mysql-instances-on-fusion-io-iodrive/>

Hardware Is Not The Only Solution

- ◆ Decrease working set (optimize, data types, archive)
- ◆ Tune OS/MySQL Configuration
- ◆ Use the right Storage Engine
- ◆ Change schema/application
- ◆ Might be if more risk adverse/enough \$\$

Choosing Hardware For MySQL

- ◆ Numbers Everybody Should Know
- ◆ CPU
- ◆ Memory
- ◆ Disk
- ◆ Network
- ◆ Amazon Cloud
- ◆ Running MySQL



Kenny Gryp
<kenny.gryp@percona.com>
@gryp

We're Hiring!
www.percona.com/about-us/careers/