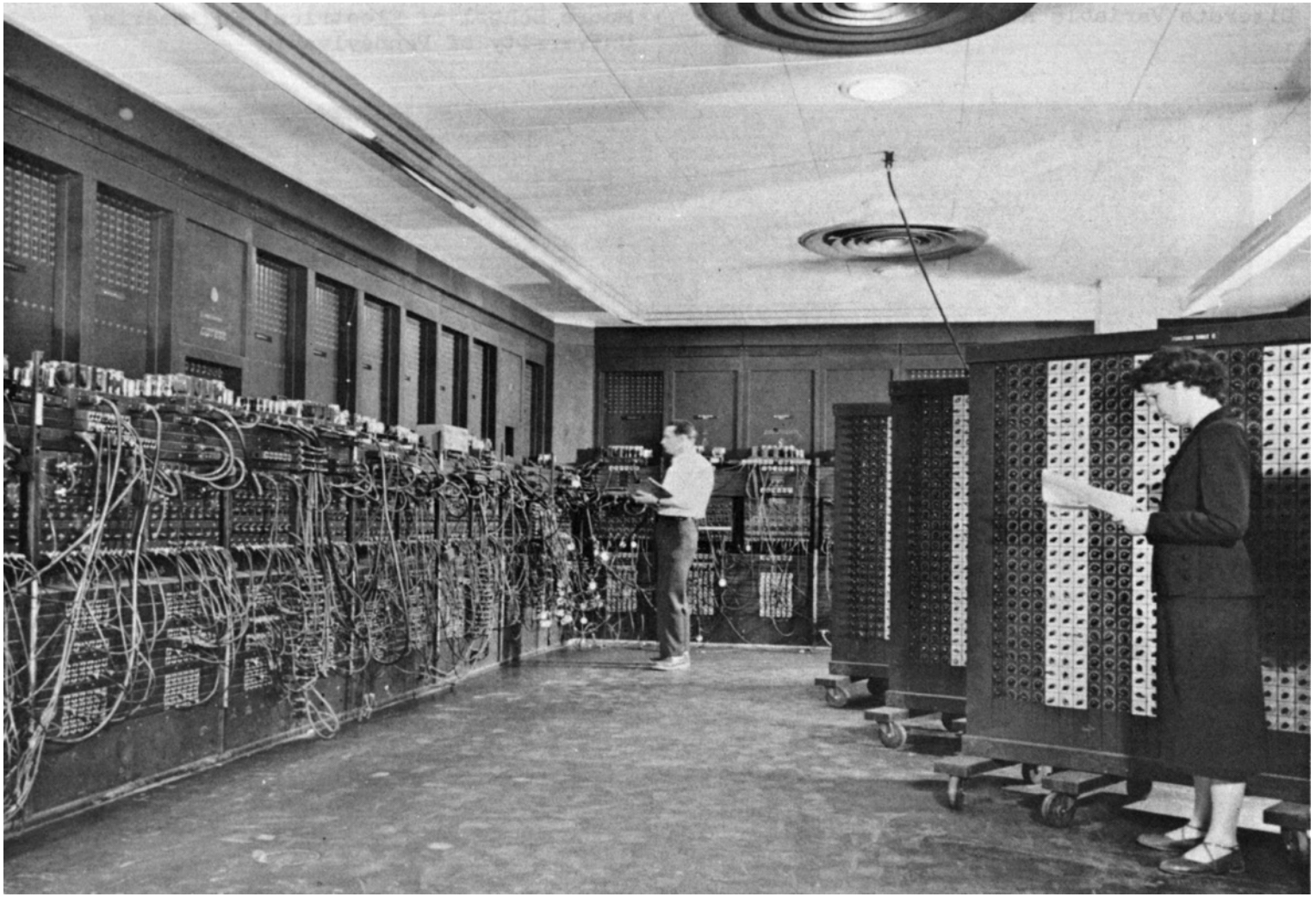
More Than Storage

Margo Seltzer
Canada 150 Research Chair in Computer Systems
University of British Columbia

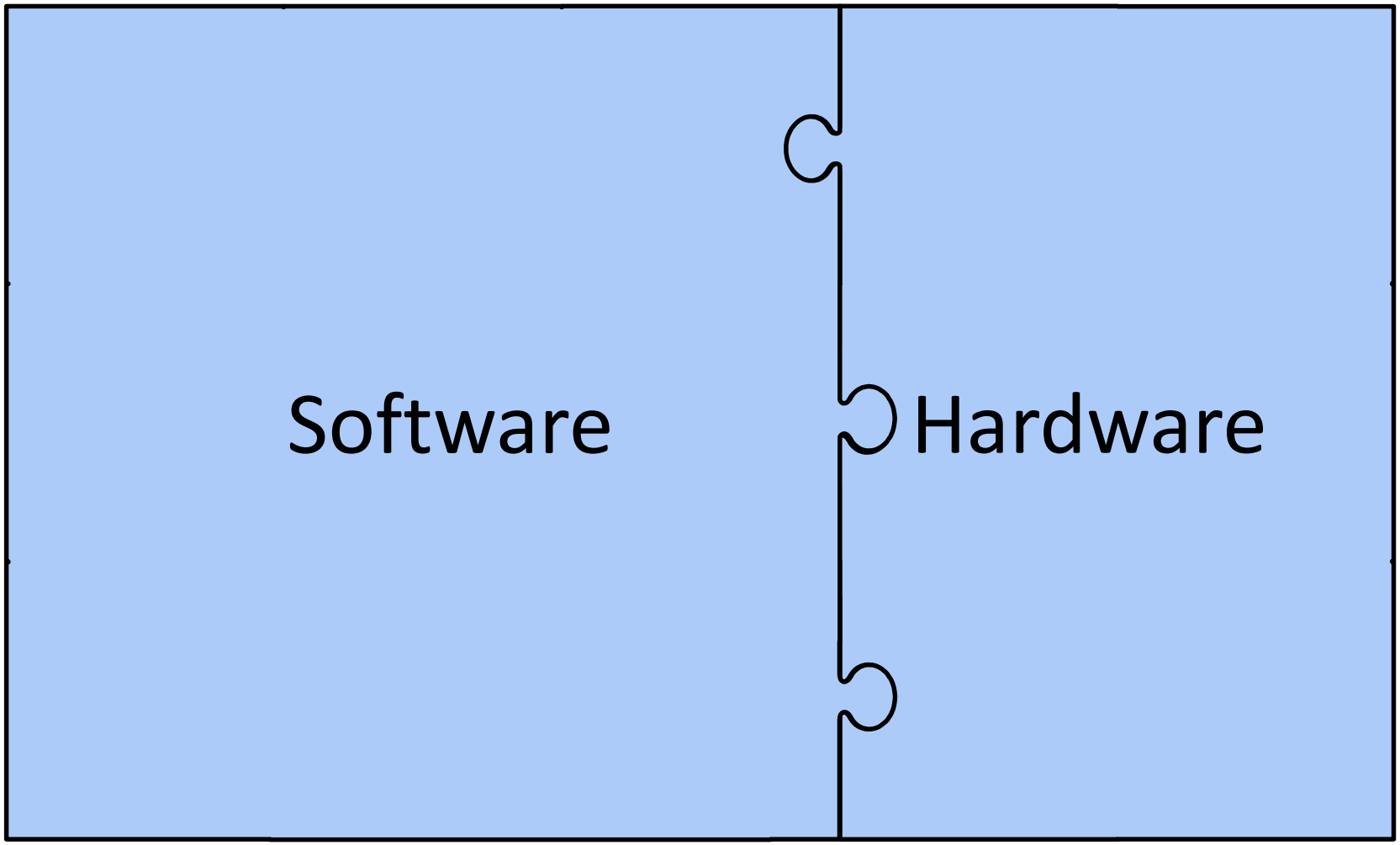
How do you build a mechanical computing device?



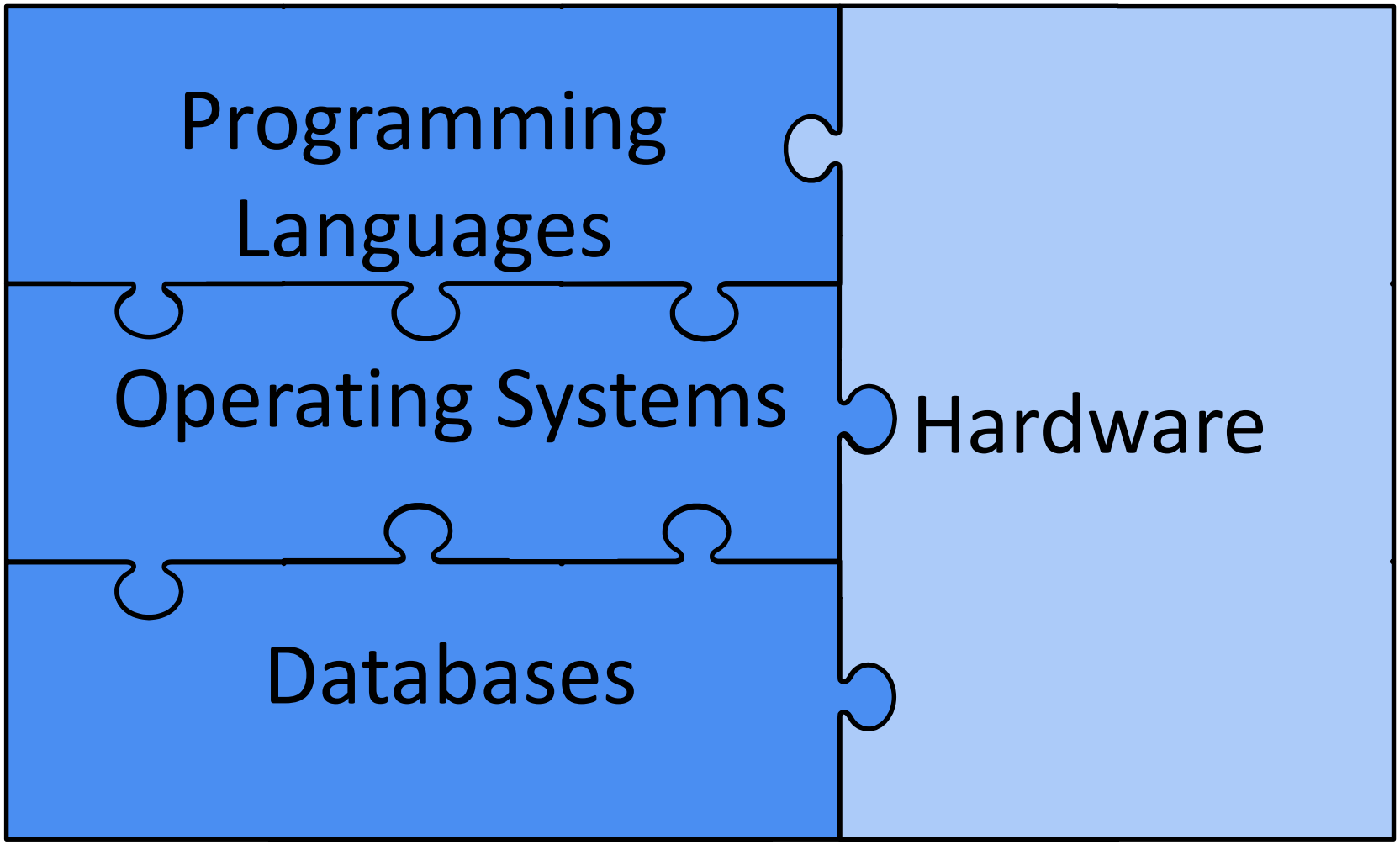
Computer Systems

Hardware, Software, and Programming

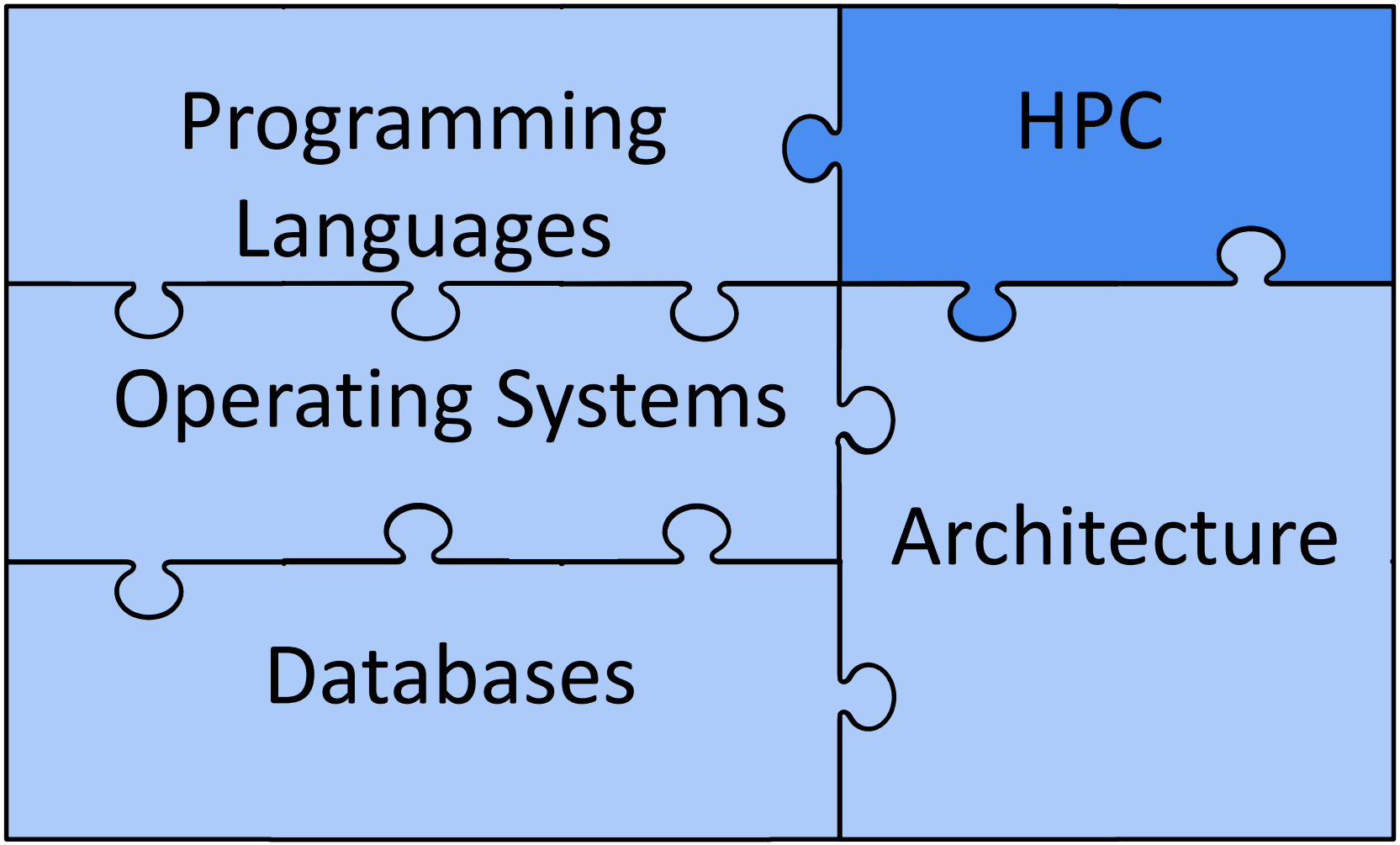
Computer Systems



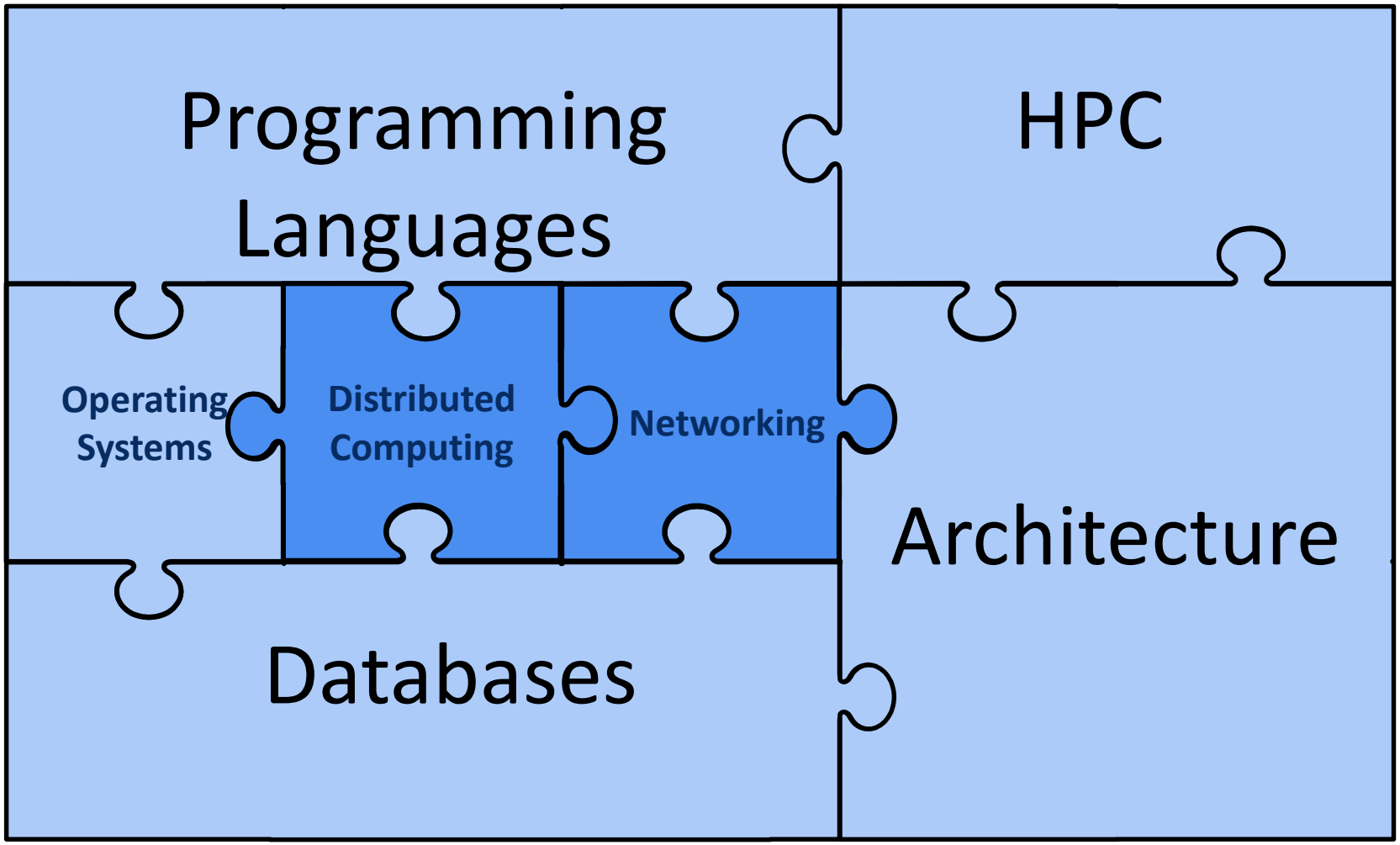
Computer Systems



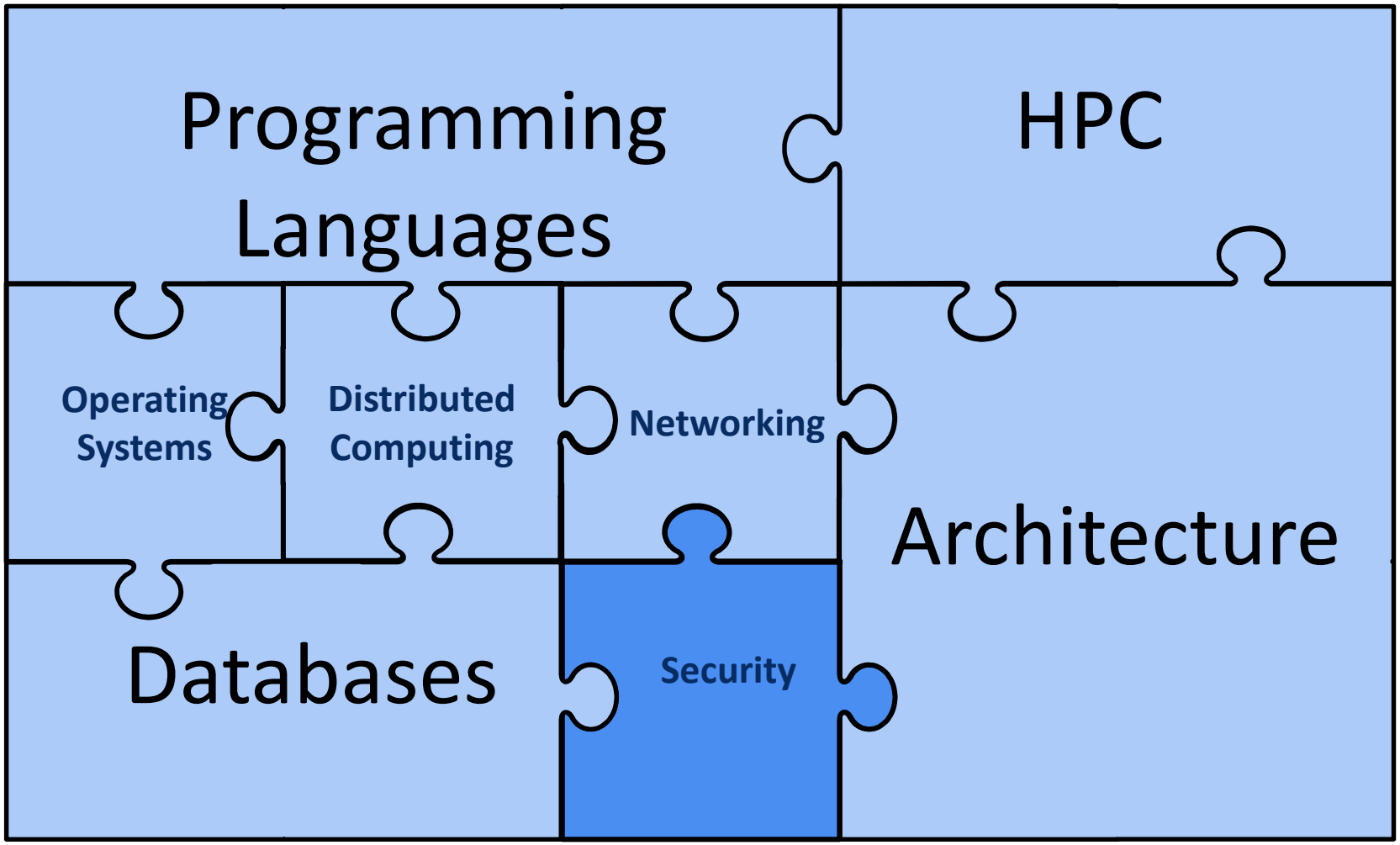
Computer Systems



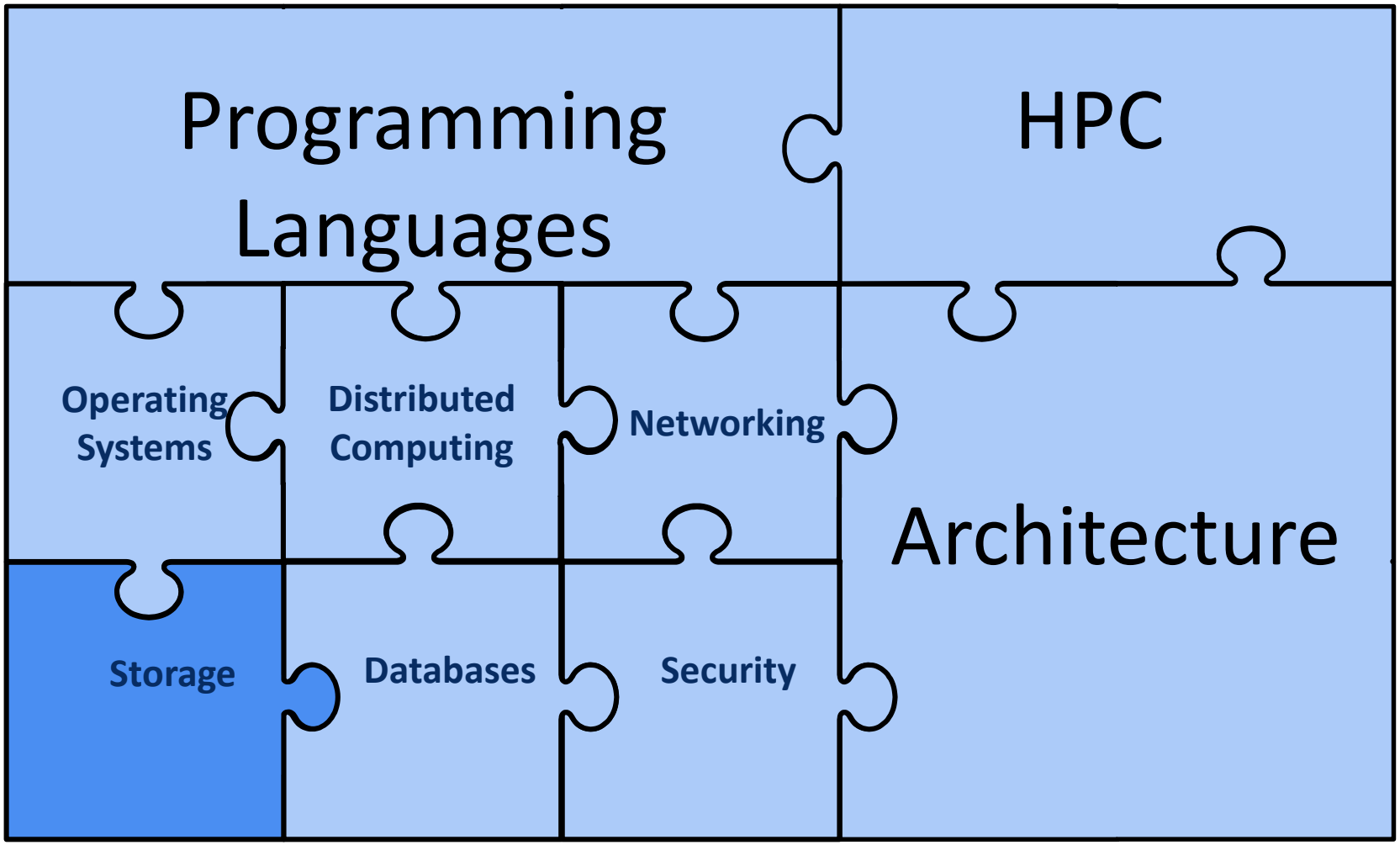
Computer Systems



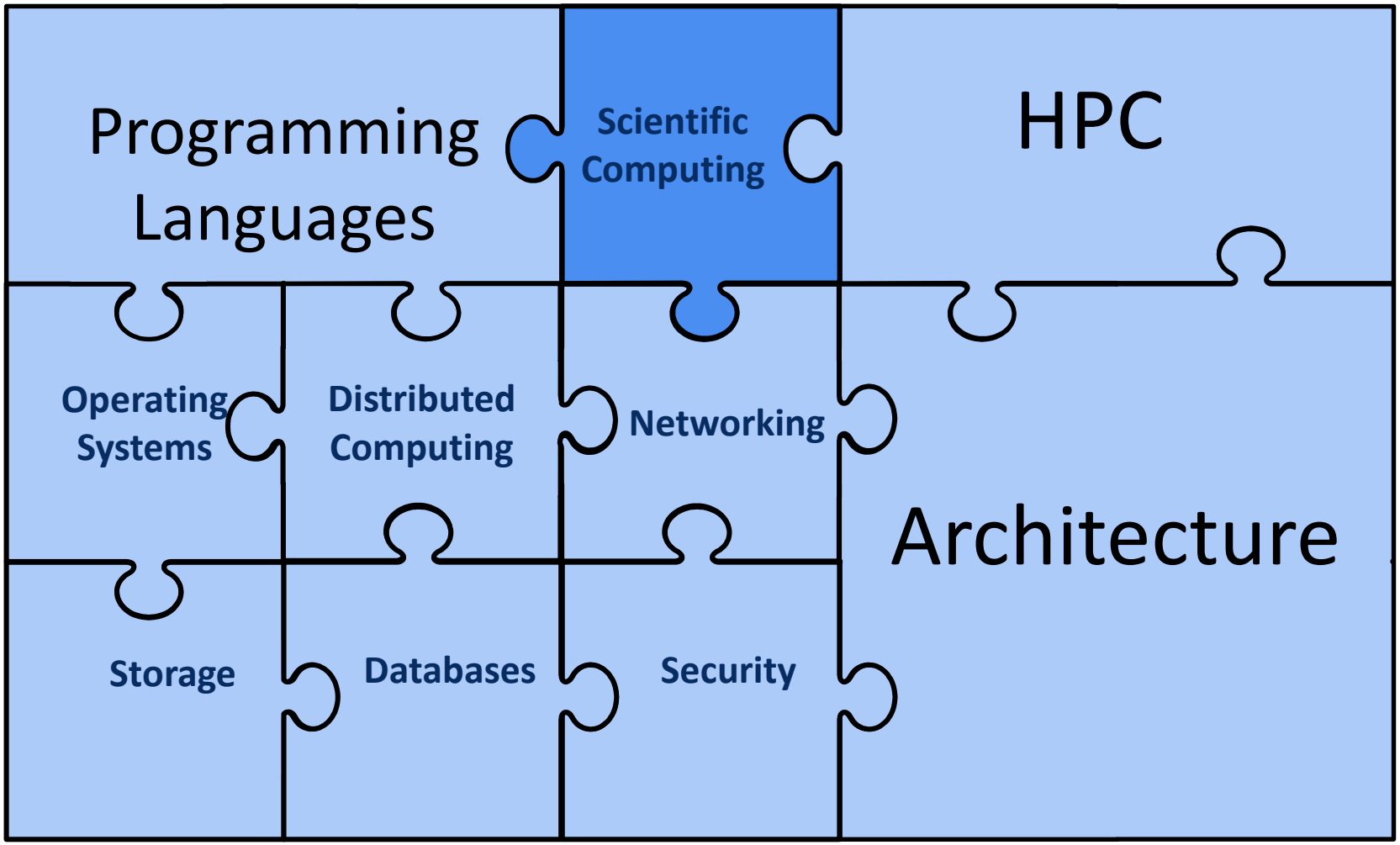
Computer Systems



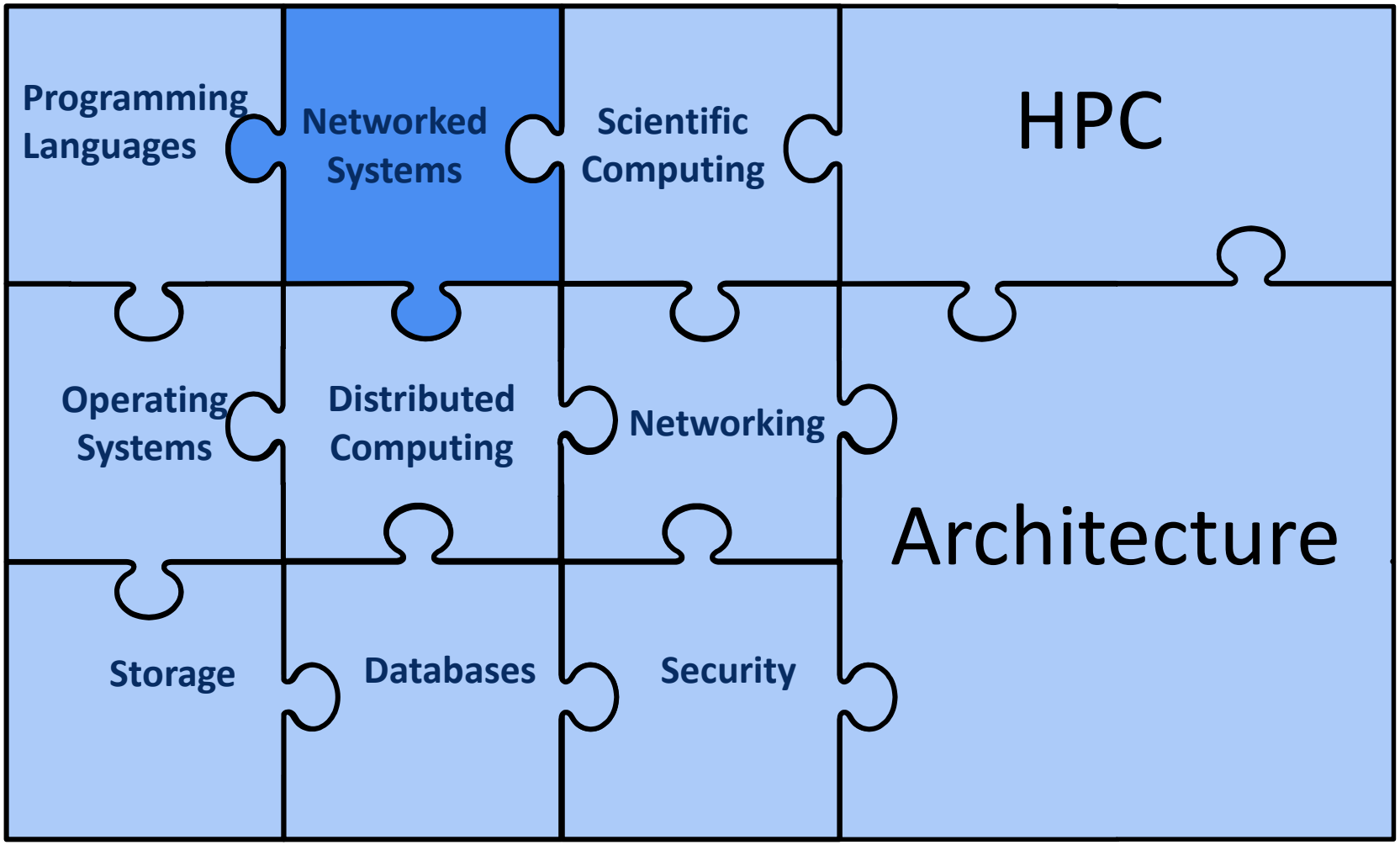
Computer Systems



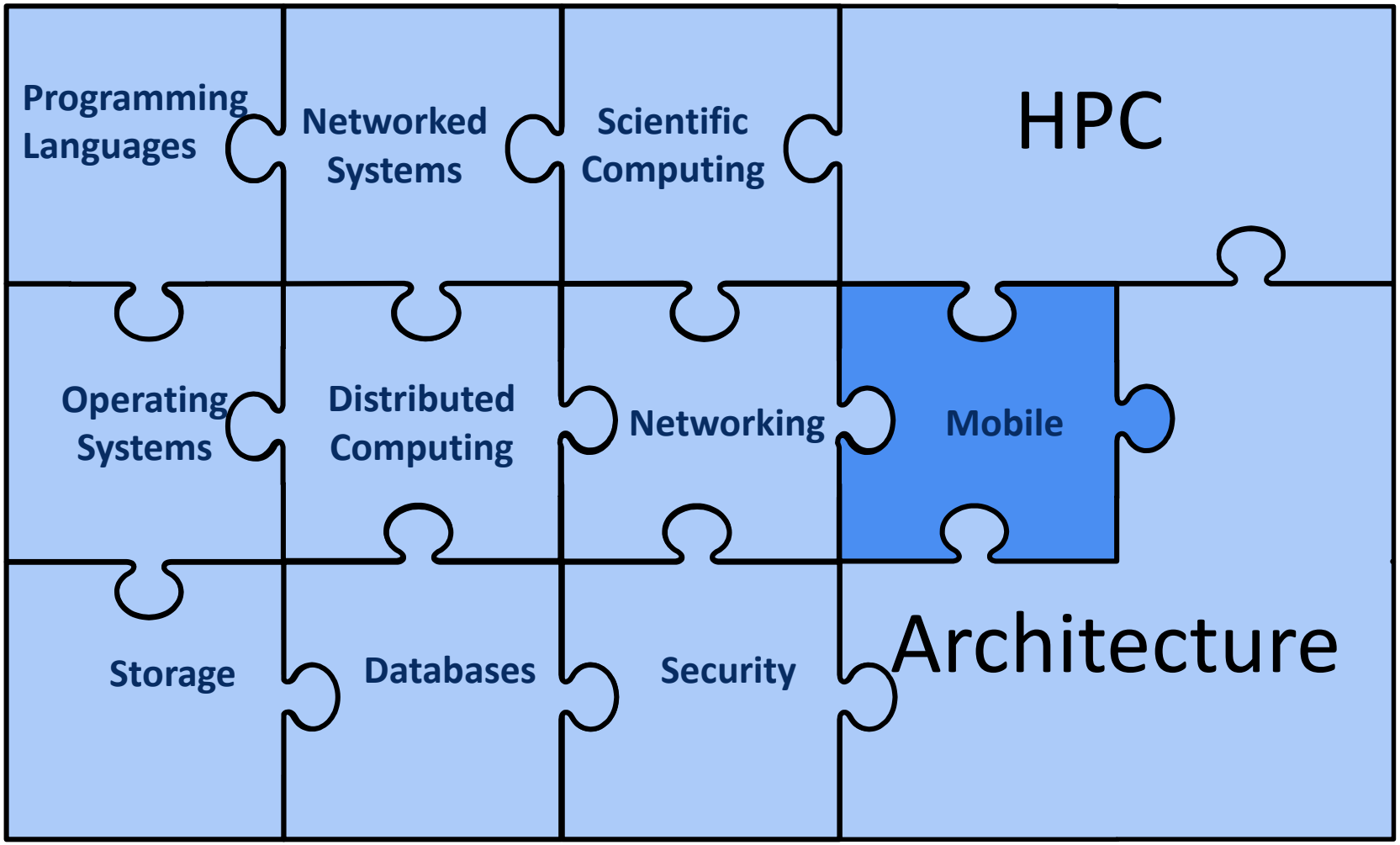
Computer Systems



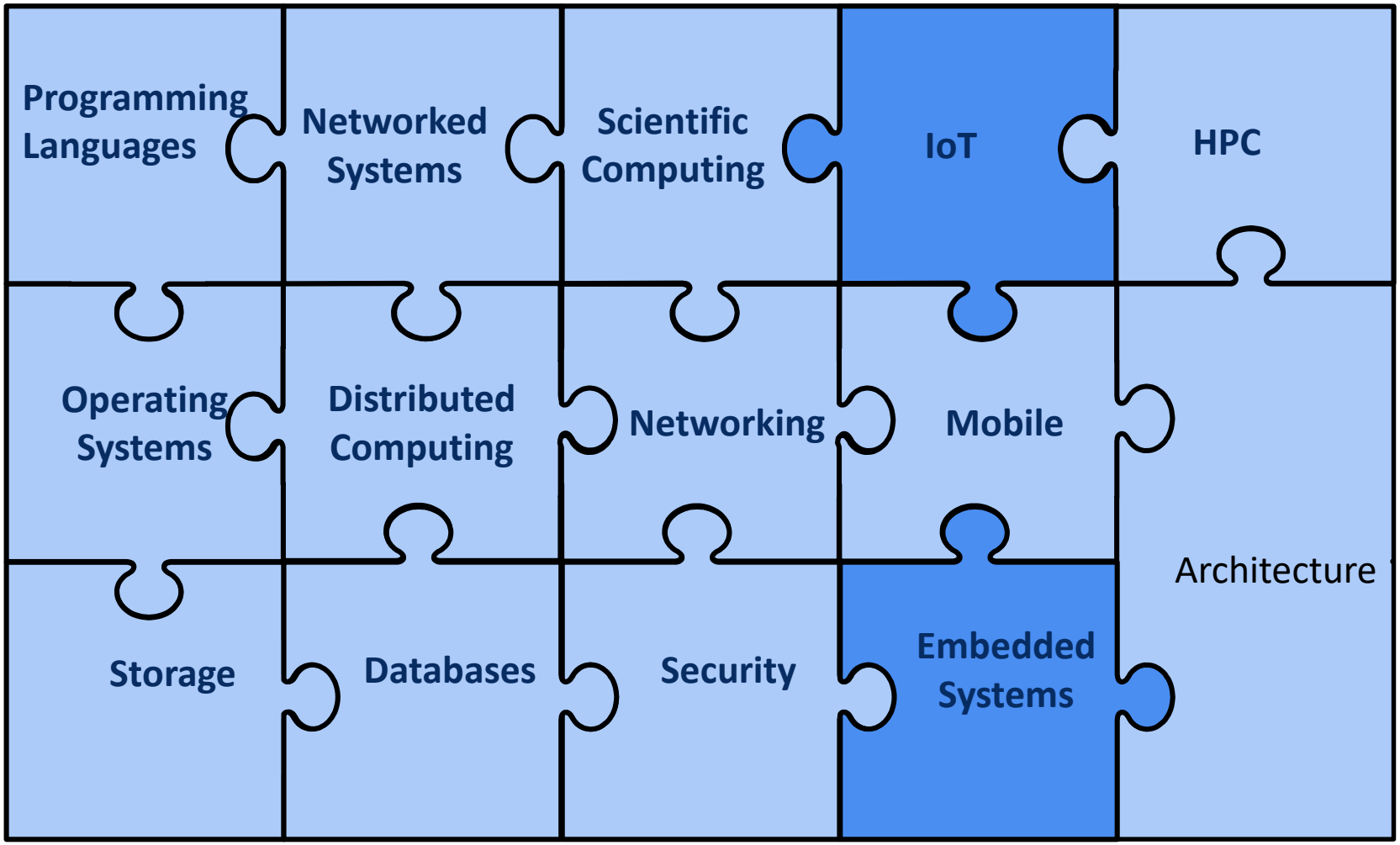
Computer Systems



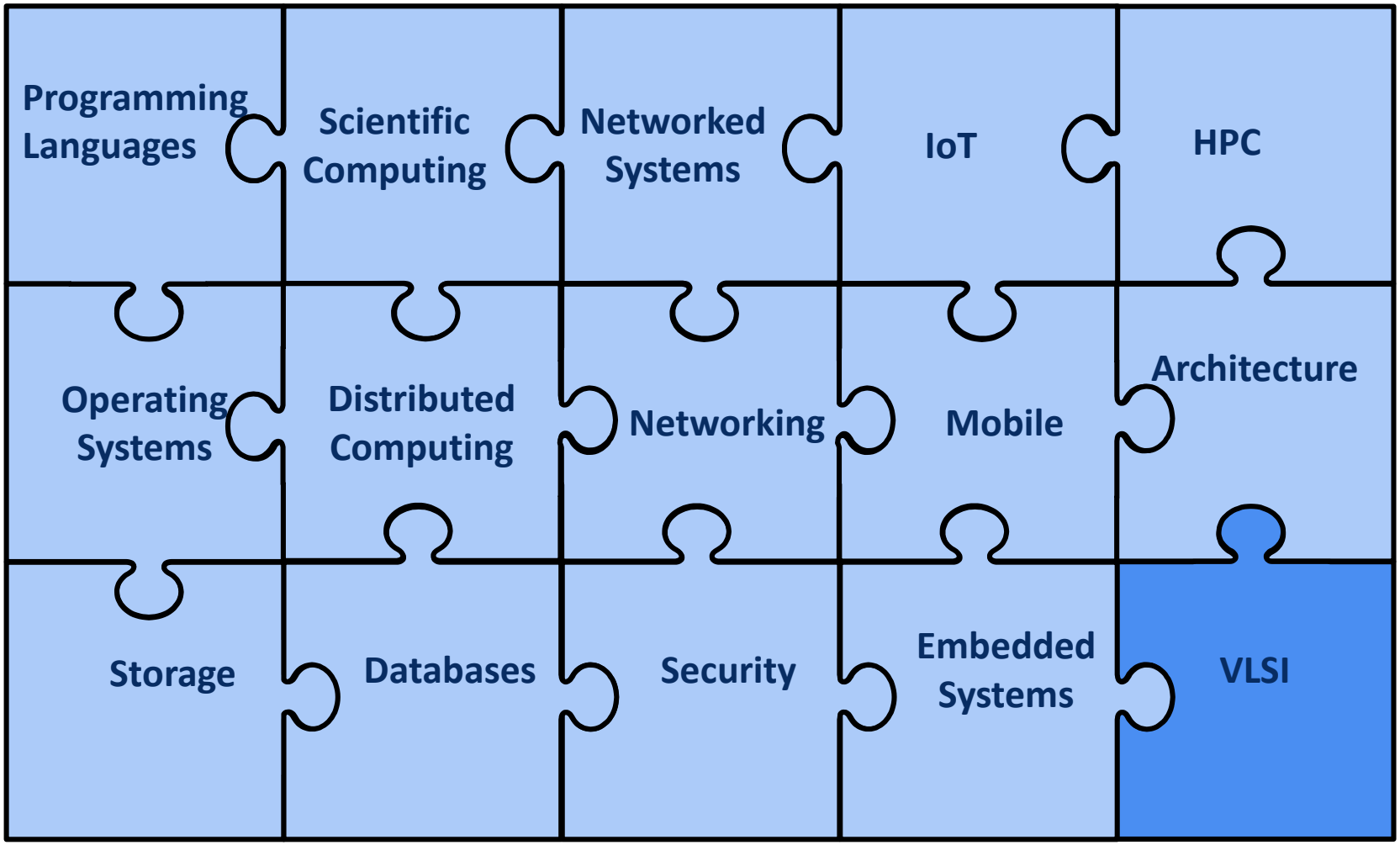
Computer Systems



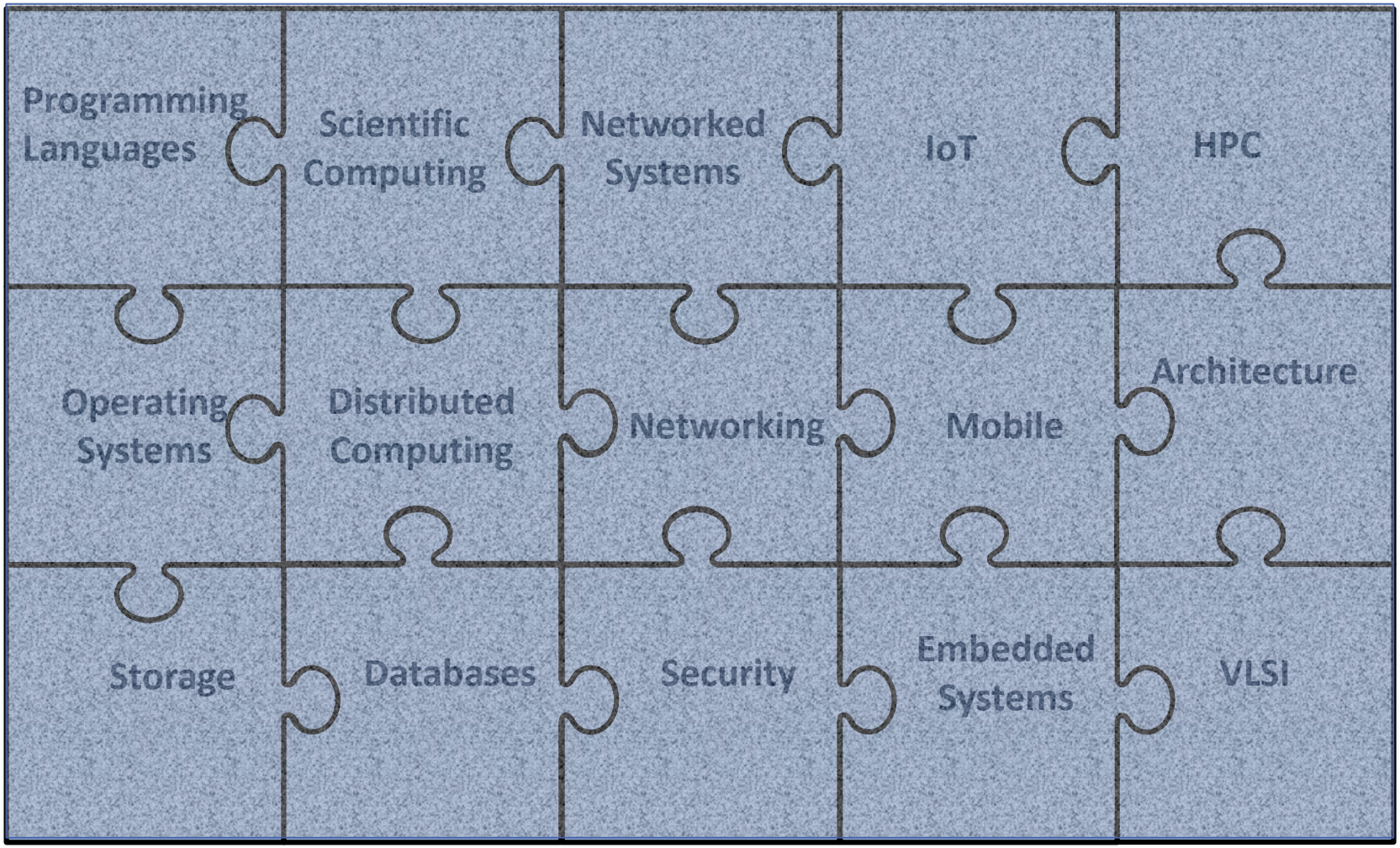
Computer Systems



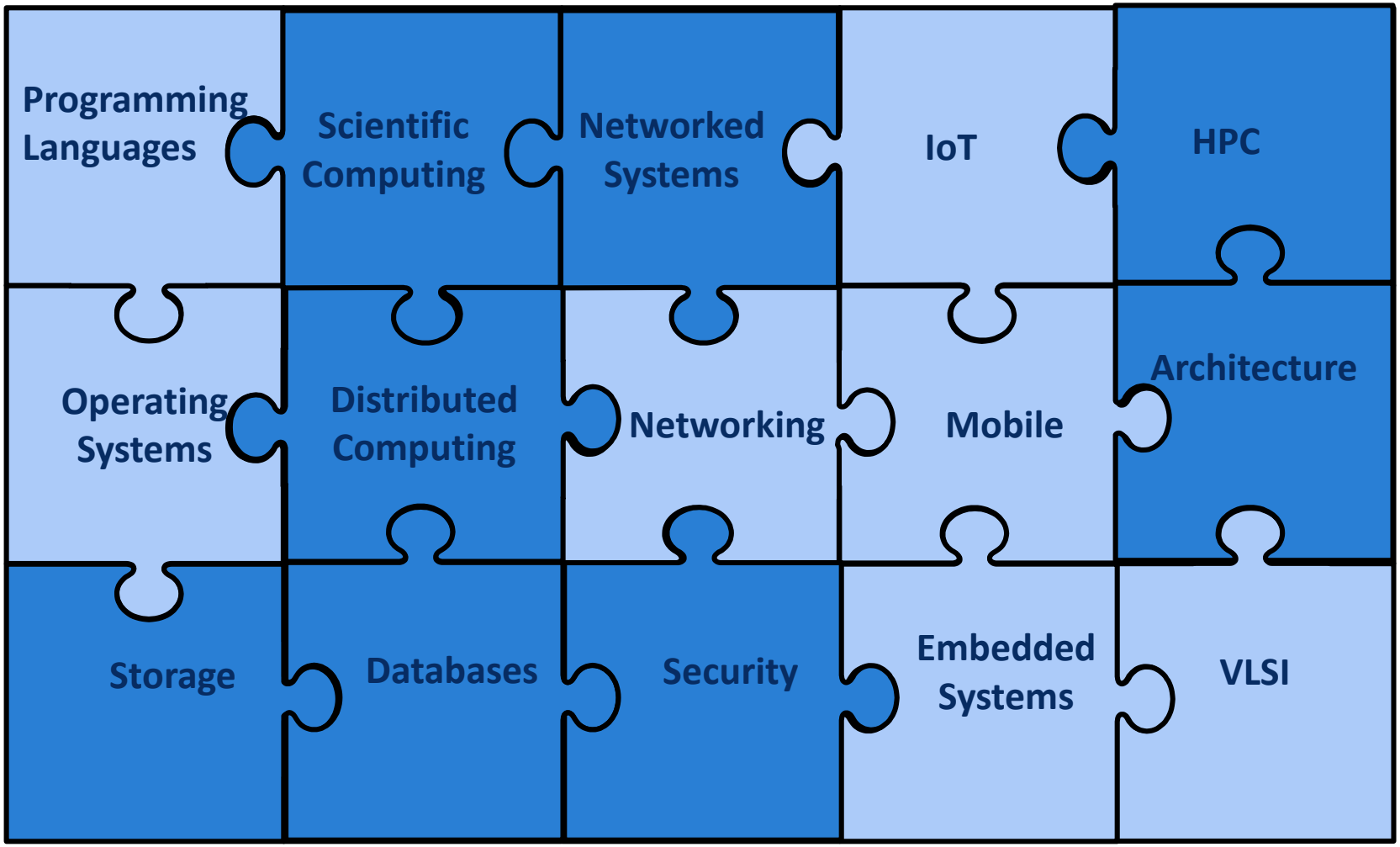
Computer Systems



Computer Systems



Storage?


Programming Languages
Scientific Computing
Networked Systems
IoT
HPC
Operating Systems
Distributed Computing
Networking
Mobile
Architecture
Storage
Databases
Security
Embedded Systems
VLSI

Three Storage Vignettes

Runtime Provenance Applications

Adapting Existing Solutions

Keys	Values
Session1	Cidon, Manno, Evans, Guyot
Session2	Blomer, Hallak, Bbrown, Manno
Session3	Strauss, Peglar, Gervasi
Sesson4	Lightning Talks

Building Custom Solutions

Keys	Values
Session1	Cidon, Manno, Evans, Guyot
Session2	Blomer, Hallak, Bbrown, Manno
Session3	Strauss, Peglar, Gervasi
Sesson4	Lightning Talks

Runtime Provenance Applications

Thomas Pasquier

Michael (Xueyuan)Han

Jean Bacon

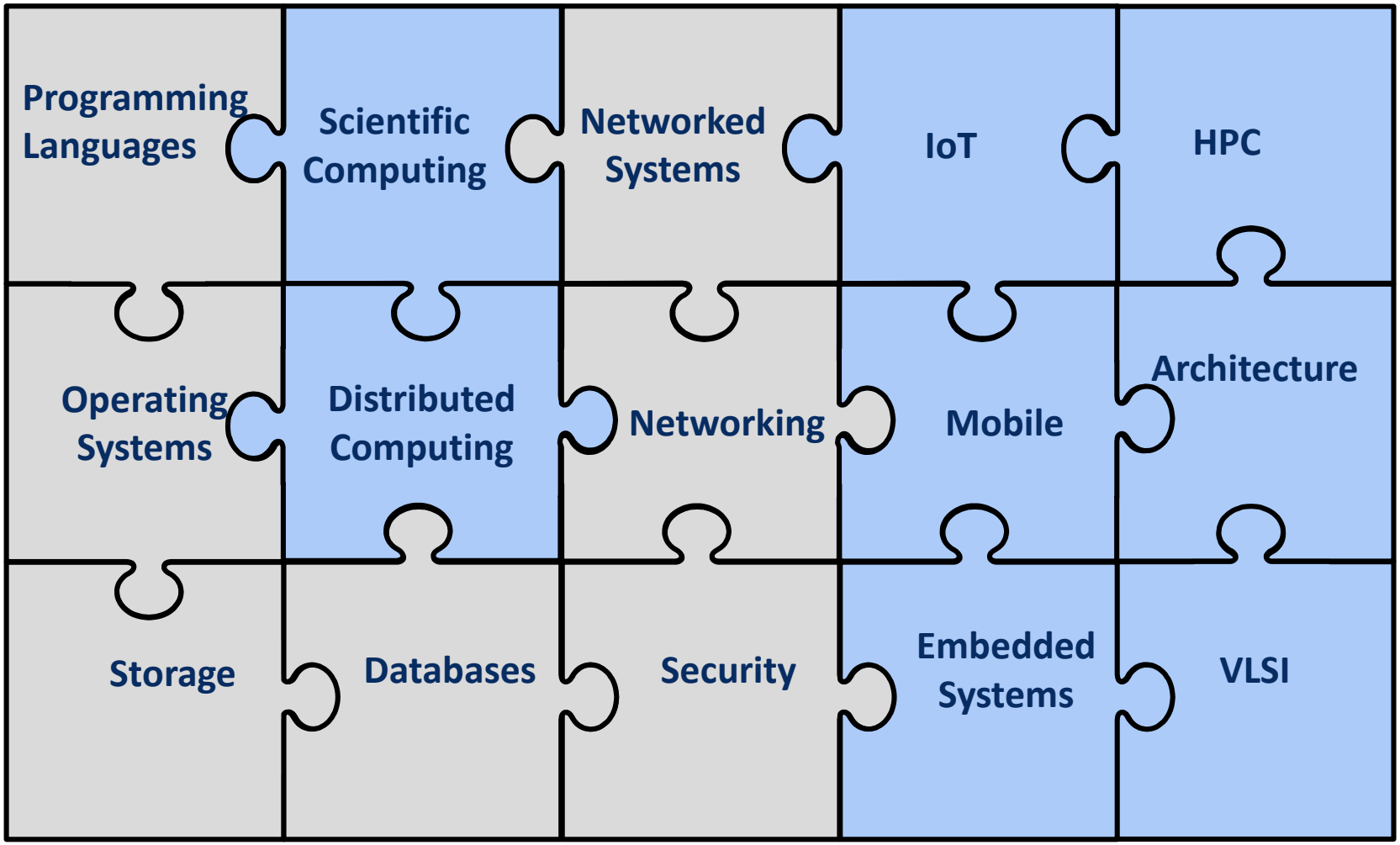
David Eyers

Oliveir Hermant

Thomas Moyer

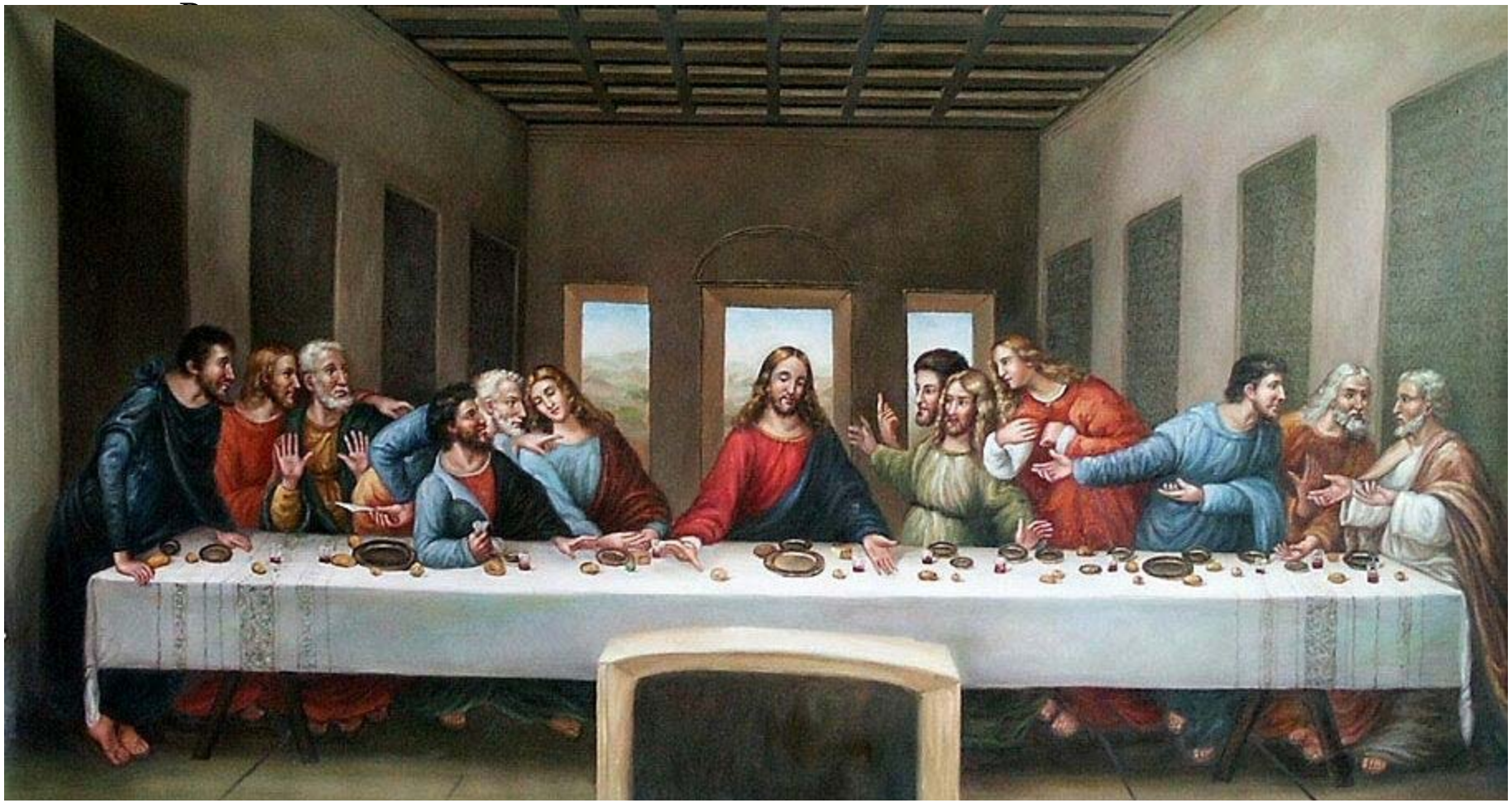
Adam Bates

Runtime Provenance Applications

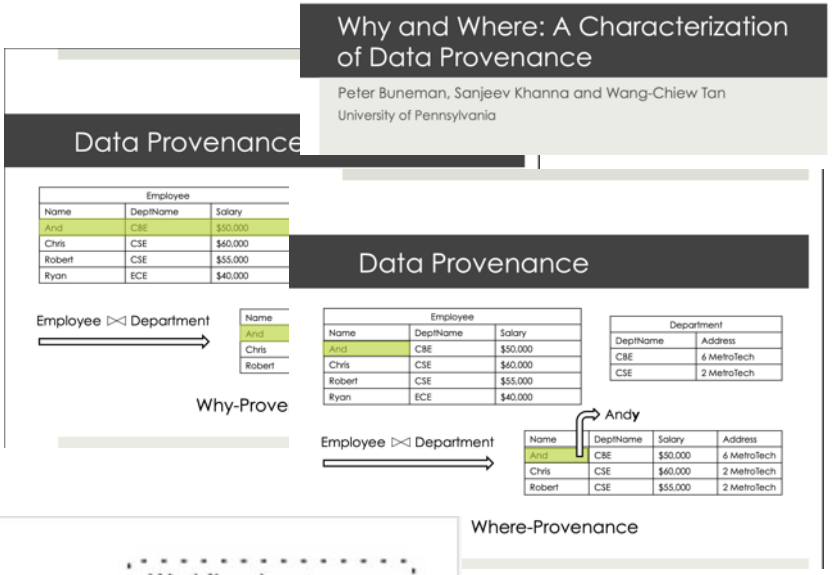


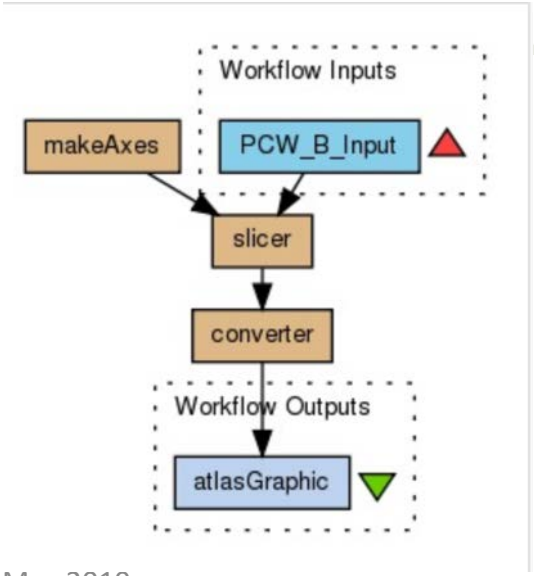
Provenance 101

Provenance 101

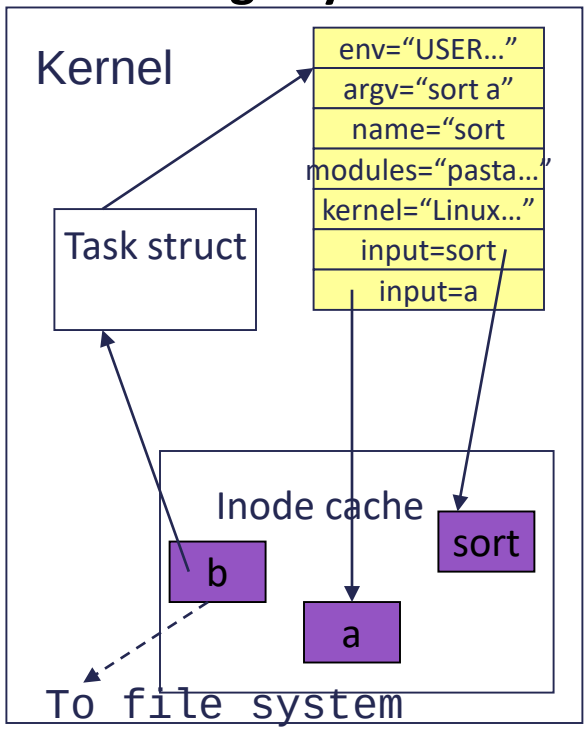


From Art to Computer Science

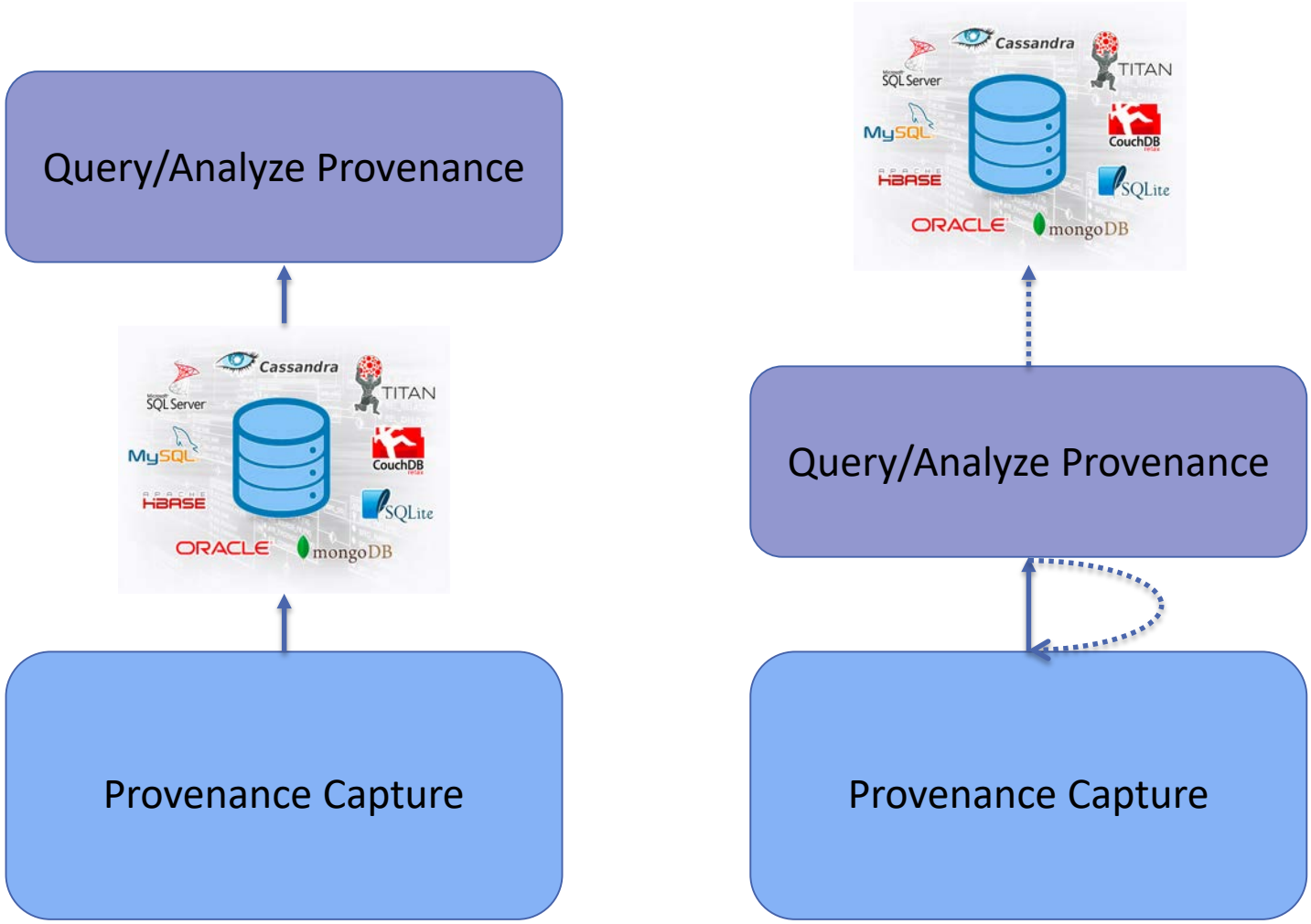




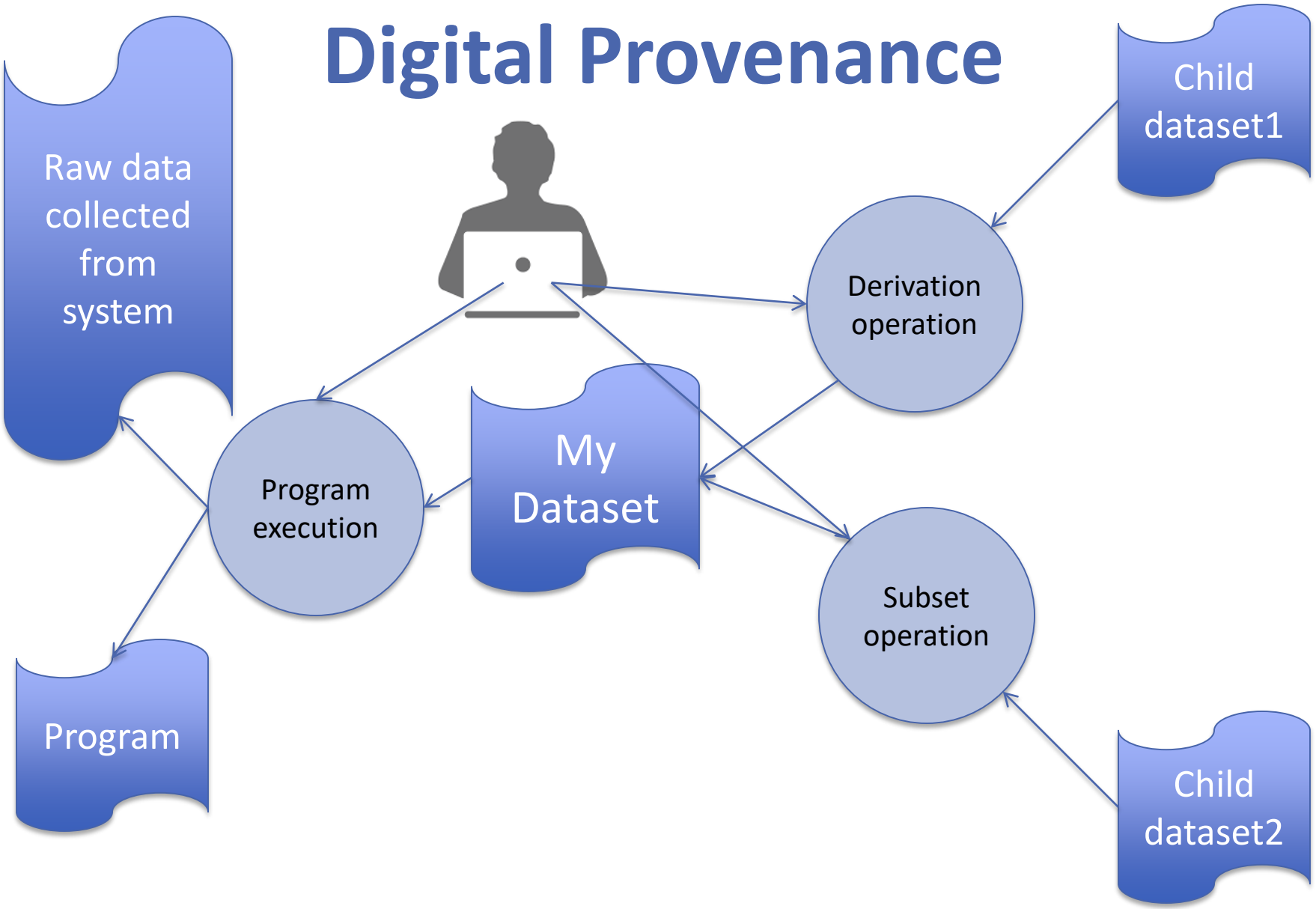
Provenance-Aware Storage Systems



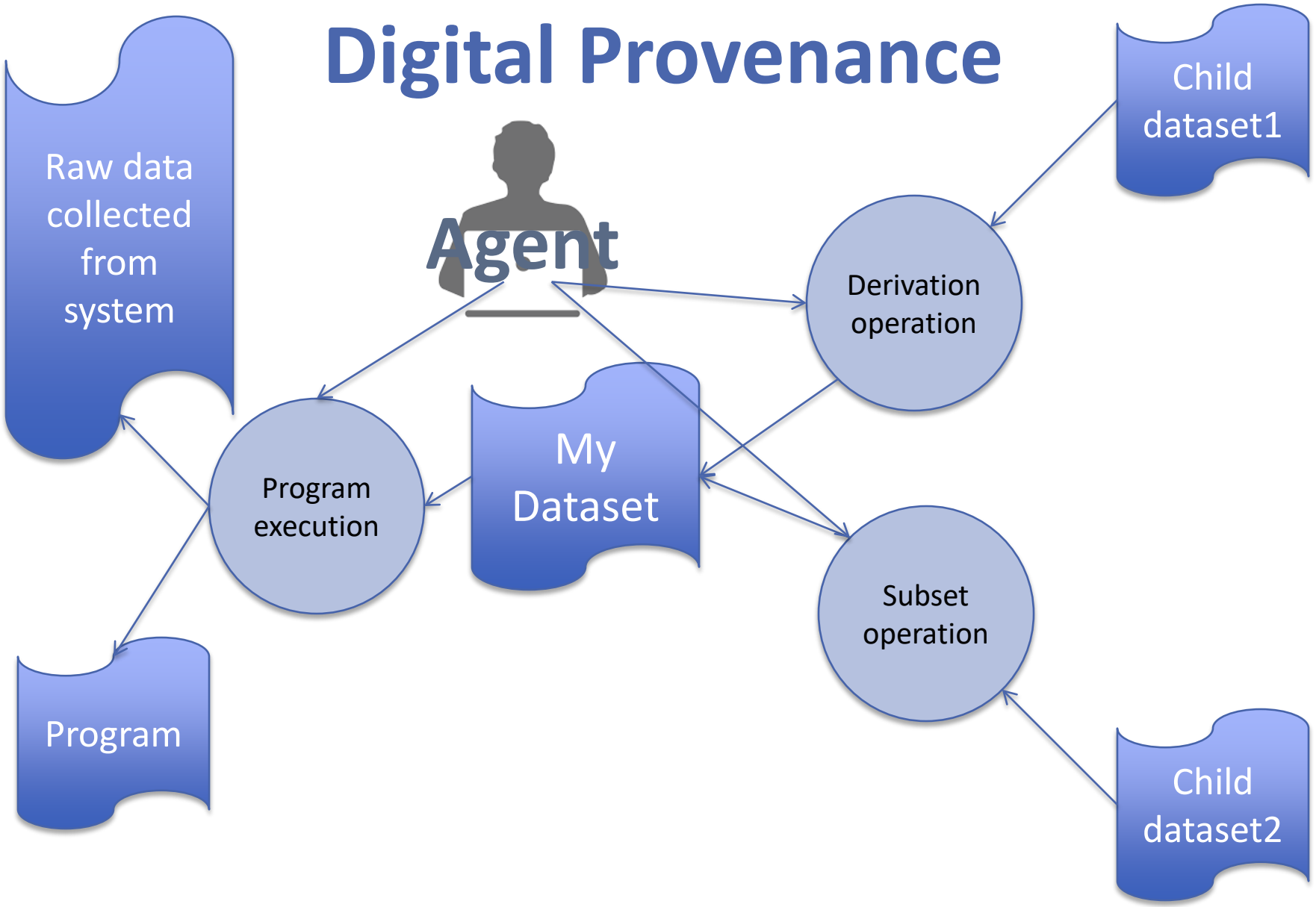
The Big Idea in one Slide



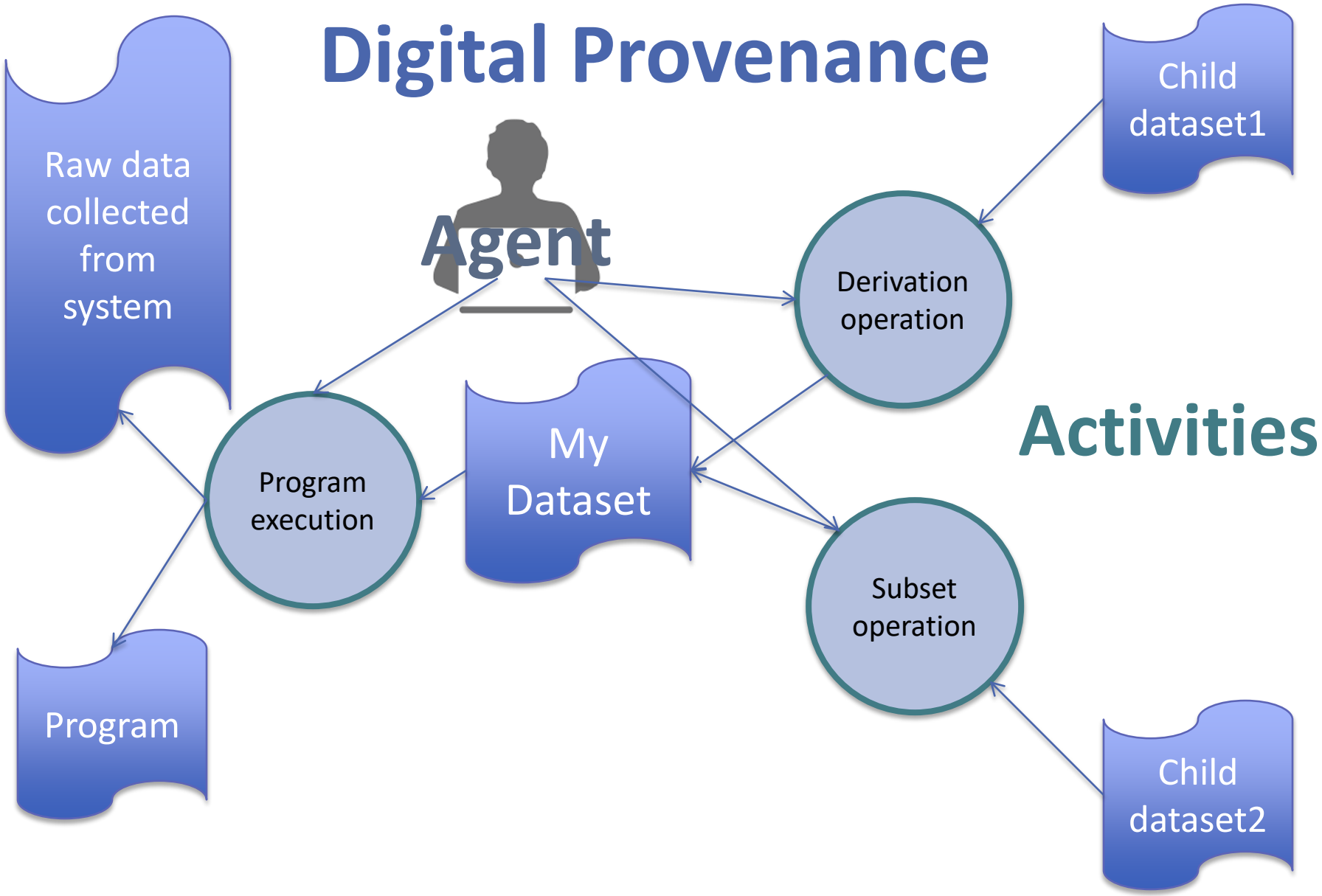
Digital Provenance

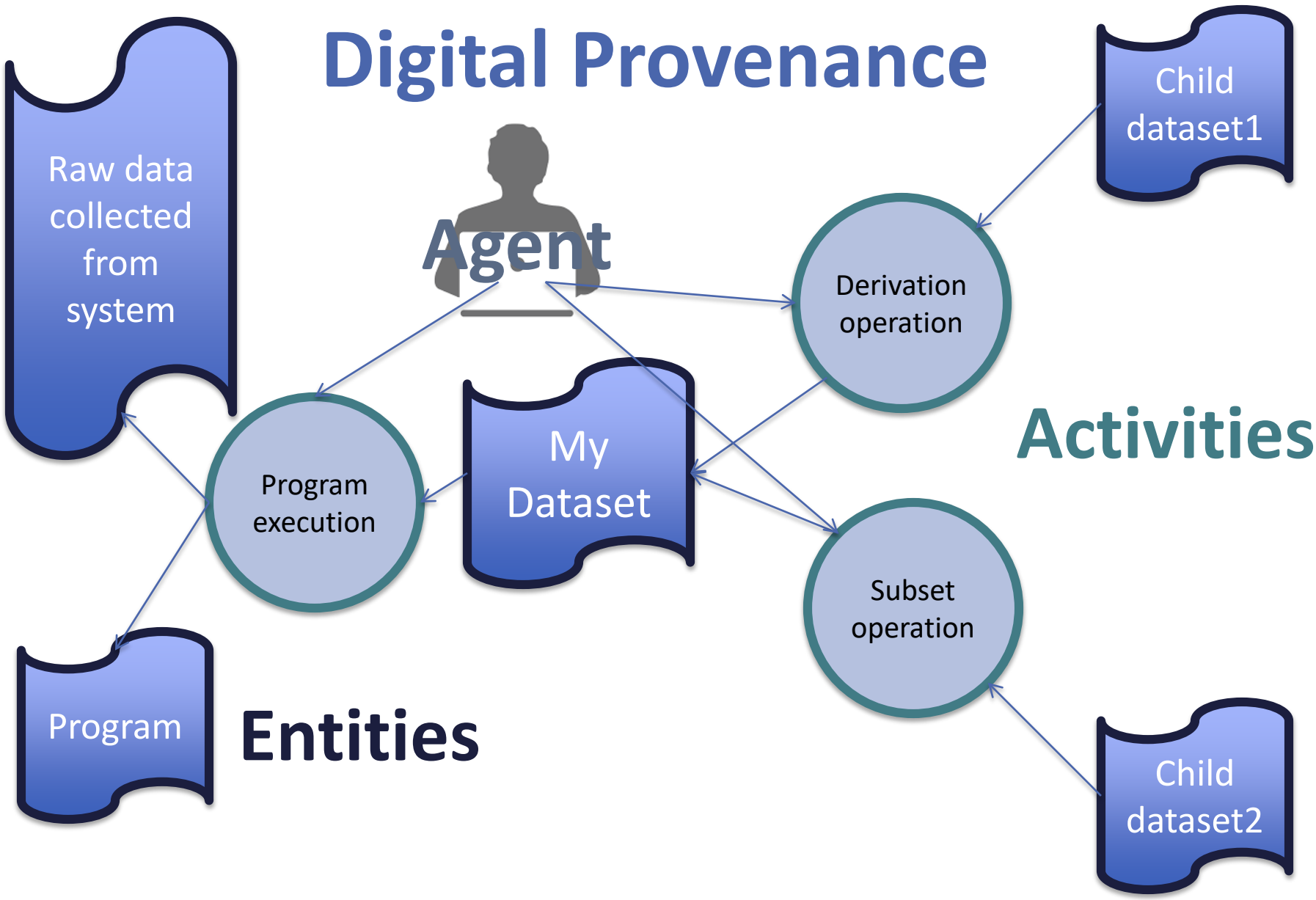


Digital Provenance

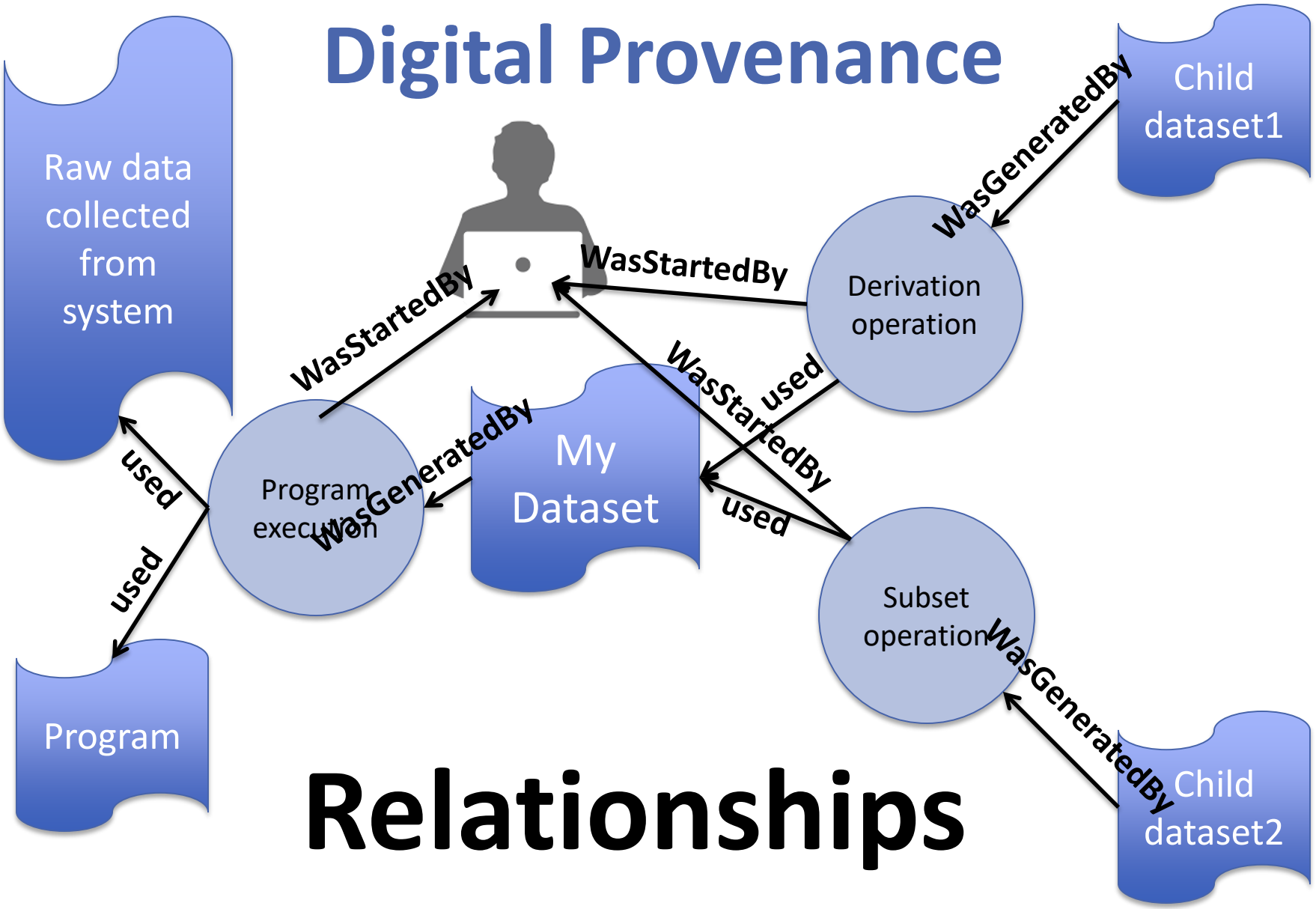


Digital Provenance



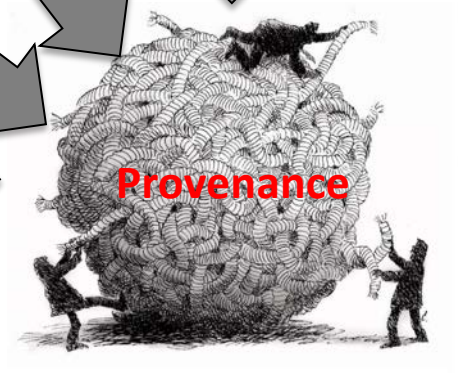

Digital Provenance
Raw data collected from system
Agent
Child dataset1
Derivation operation
Activities
Program execution
My Dataset
Subset operation
Program
Entities
Child dataset2

Digital Provenance

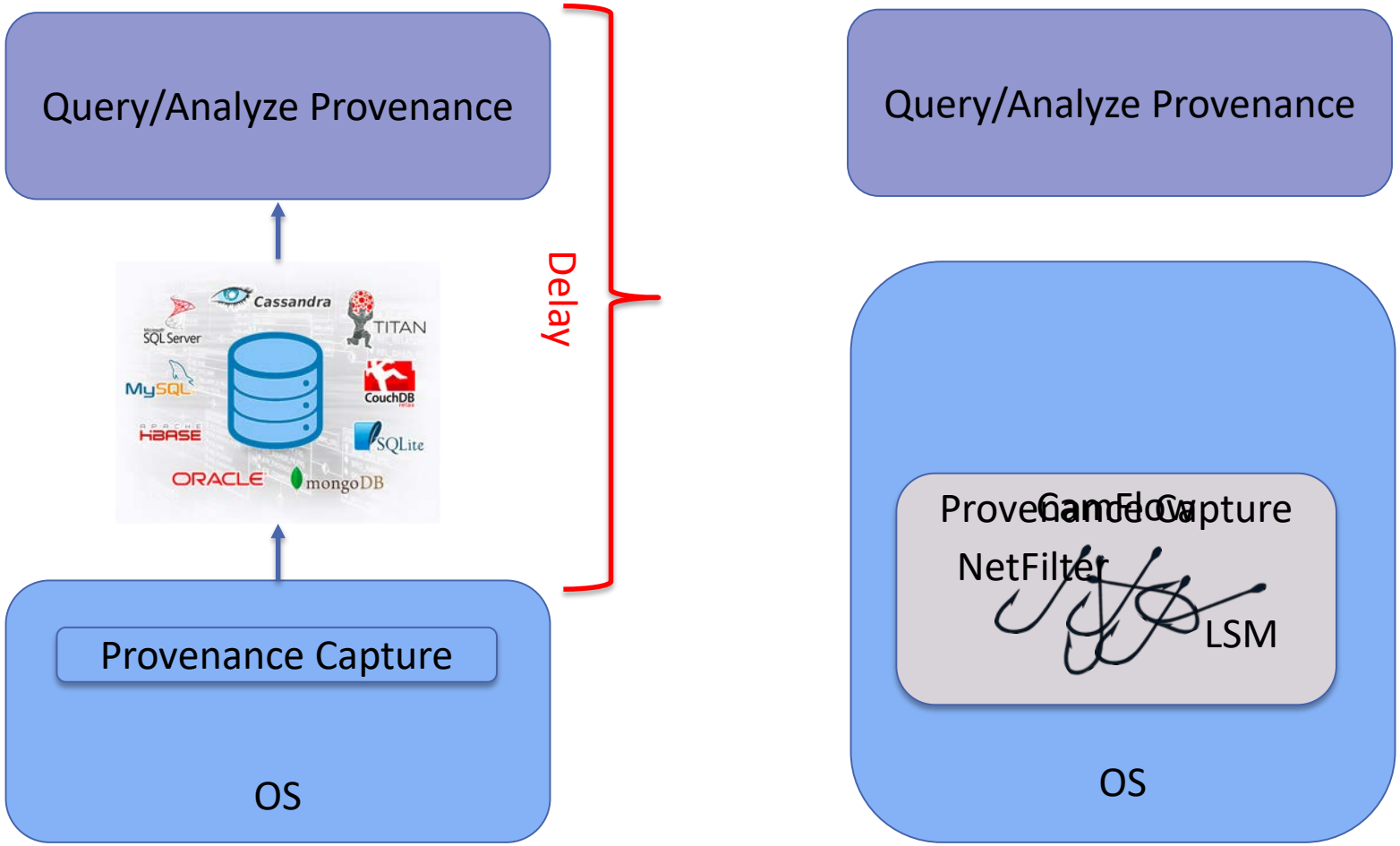


Relationships

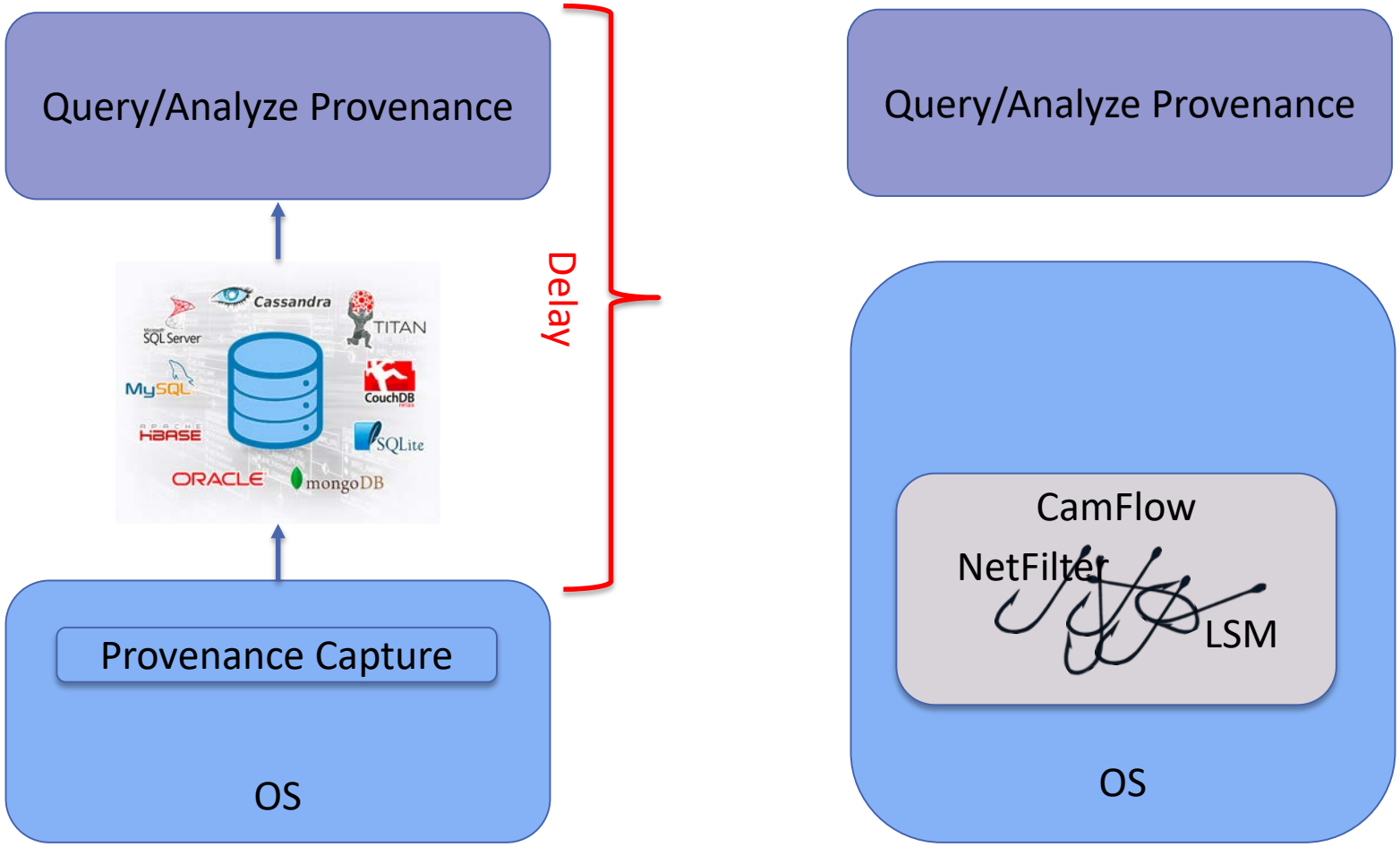
All Kinds of Provenance



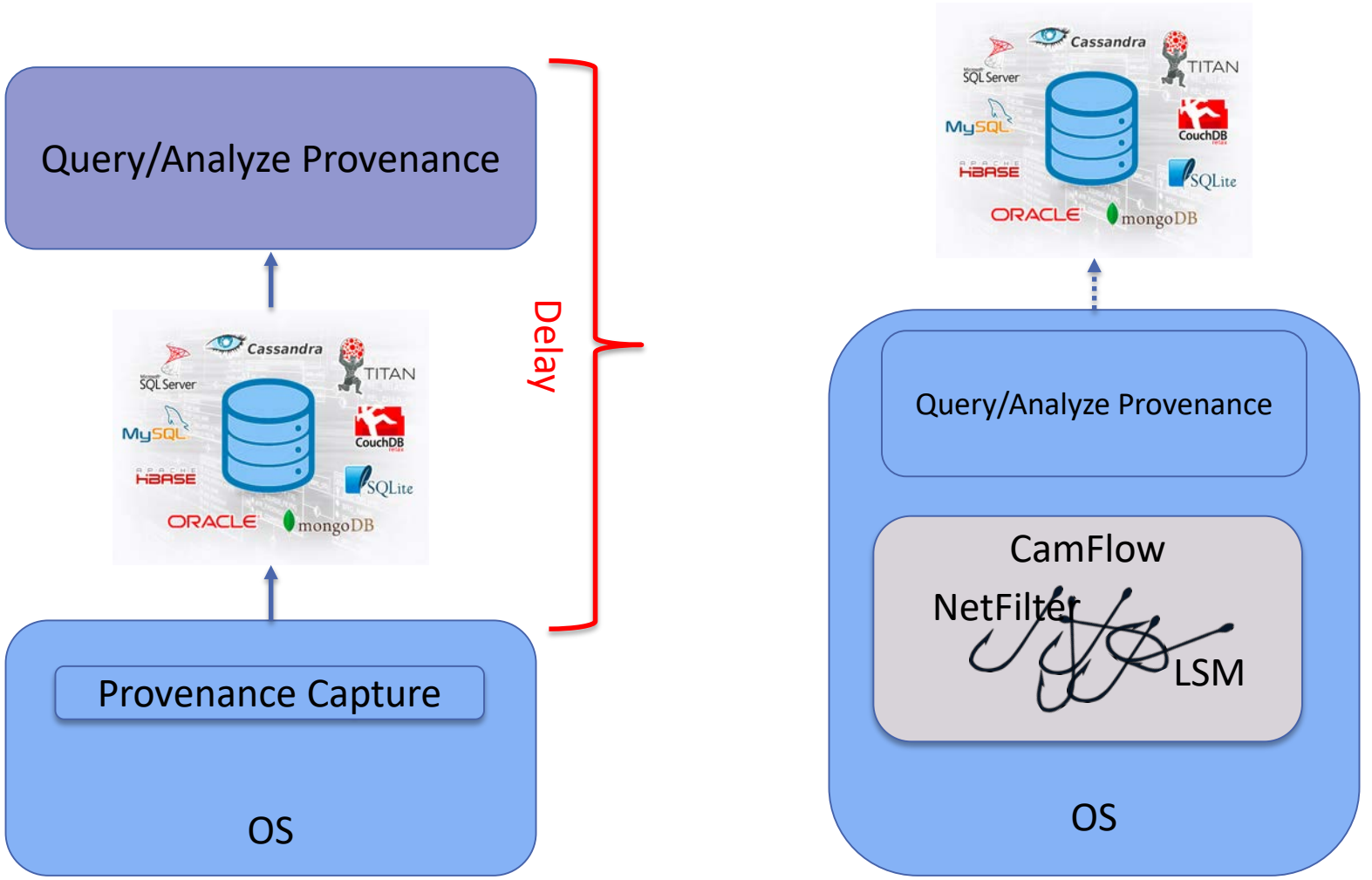
CamQuery Architecture



CamQuery Architecture



CamQuery Architecture

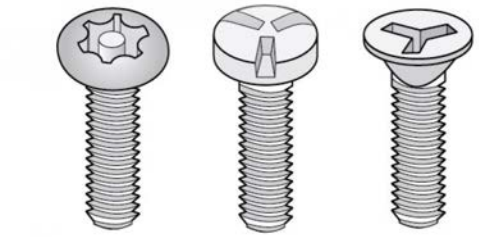


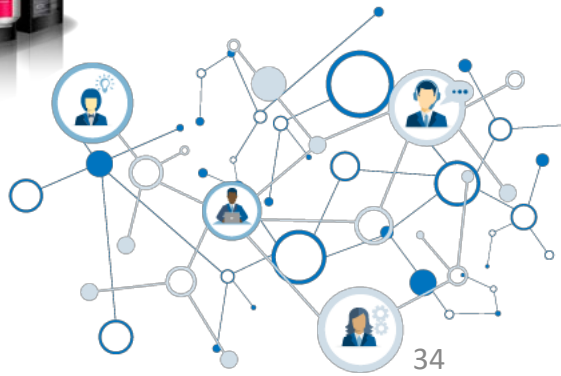
Architectural Implications

Conventional Provenance Applications		CamQuery Provenance Applications
Static Graph Analysis	→	Streaming Graph Analysis
Detection	→	Prevention
F(Graph)	→	F(Function)
Immutable	→	Mutable

Sample Applications

Information Flow





Writing an Application

```
#define KERNEL_QUERY
#include “include/camquery.h”

static label_t confidential;

static void init(void)
     confidential =
         get_label(“confidential”);
}
```

```
static int
in_edge(union prov_msg *edge,
  union prov_msg * node)
{
     if (has_label(edge, confidential)) {
          add_label(node, confidential);
          if (node_type(node) == ENT_INODE_SOCKET)
               return PROV_RAISE_WARNING;
     }
     return 0;
}
```

```
static int out_edge(union prov_msg *node,
  union prov_msg *edge) {
     switch (edge_type(edge)) {
          case RL_WRITE:
          case RL_READ:
          case RL_SND:
          case RL_RCV:
          case RL_VERSION:
          case RL_VERSION_PROCESS:
          case RL_CLONE:
               if (has_label(node, confidential))
                    add_label(edge, confidential);
     }
     return 0;
}
```

```
QUERY_NAME(“Propagate labels”);
QUERY_DESCRIPTION(“Example query”);
QUERY_AUTHOR(“Not me.”);
QUERY_VERSION(“0.1”);
QUERY_LICENSE(“GPL”);
register_query(init, in_edge, out_edge);
```

Writing an Application

```
#define KERNEL_QUERY
#include “include/camquery.h”

static label_t confidential;

static void init(void)
     confidential =
         get_label(“confidential”);
}
```

```
static int
in_edge(union prov_msg *edge,
  union prov_msg * node)
{
     if (has_label(edge, confidential)) {
          add_label(node, confidential);
          if (node_type(node) == ENT_INODE_SOCKET)
               return PROV_RAISE_WARNING;
     }
     return 0;
}
```

```
static int out_edge(union prov_msg *node,
  union prov_msg *edge) {
     switch (edge_type(edge)) {
          case RL_WRITE:
          case RL_READ:
          case RL_SND:
          case RL_RCV:
          case RL_VERSION:
          case RL_VERSION_PROCESS:
          case RL_CLONE:
               if (has_label(node, confidential))
                    add_label(edge, confidential);
     }
     return 0;
}
```

```
QUERY_NAME(“Propagate labels”);
QUERY_DESCRIPTION(“Example query”);
QUERY_AUTHOR(“Not me.”);
QUERY_VERSION(“0.1”);
QUERY_LICENSE(“GPL”);
register_query(init, in_edge, out_edge);
```

Writing an Application

```
#define KERNEL_QUERY
#include “include/camquery.h”

static label_t confidential;

static void init(void)
     confidential =
          get_label(“confidential”);
}
```

```
static int
in_edge(union prov_msg *edge,
  union prov_msg * node)
{
     if (has_label(edge, confidential)) {
          add_label(node, confidential);
          if (node_type(node) == ENT_INODE_SOCKET)
               return PROV_RAISE_WARNING;
     }
     return 0;
}
```

```
static int out_edge(union prov_msg *node,
  union prov_msg *edge) {
     switch (edge_type(edge)) {
          case RL_WRITE:
          case RL_READ:
          case RL_SND:
          case RL_RCV:
          case RL_VERSION:
          case RL_VERSION_PROCESS:
          case RL_CLONE:
               if (has_label(node, confidential))
                    add_label(edge, confidential);
     }
     return 0;
}
```

```
QUERY_NAME(“Propagate labels”);
QUERY_DESCRIPTION(“Example query”);
QUERY_AUTHOR(“Not me.”);
QUERY_VERSION(“0.1”);
QUERY_LICENSE(“GPL”);
register_query(init, in_edge, out_edge);
```

Writing an Application

```
#define KERNEL_QUERY
#include “include/camquery.h”

static label_t confidential;

static void init(void)
     confidential =
         get_label(“confidential”);
}
```

```
static int
in_edge(union prov_msg *edge,
  union prov_msg * node)
{
     if (has_label(edge, confidential)) {
          add_label(node, confidential);
          if (node_type(node) == ENT_INODE_SOCKET)
               return PROV_RAISE_WARNING;
     }
     return 0;
}
```

```
static int out_edge(union prov_msg *node,
  union prov_msg *edge) {
     switch (edge_type(edge)) {
          case RL_WRITE:
          case RL_READ:
          case RL_SND:
          case RL_RCV:
          case RL_VERSION:
          case RL_VERSION_PROCESS:
          case RL_CLONE:
               if (has_label(node, confidential))
                    add_label(edge, confidential);
     }
     return 0;
}
```

```
QUERY_NAME(“Propagate labels”);
QUERY_DESCRIPTION(“Example query”);
QUERY_AUTHOR(“Not me.”);
QUERY_VERSION(“0.1”);
QUERY_LICENSE(“GPL”);
register_query(init, in_edge, out_edge);
```

Writing an Application

```
#define KERNEL_QUERY
#include “include/camquery.h”

static label_t confidential;

static void init(void)
     confidential =
         get_label(“confidential”);
}
```

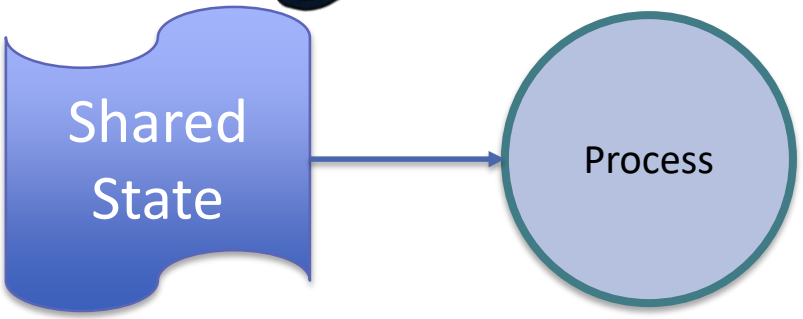
```
static int
in_edge(union prov_msg *edge,
  union prov_msg * node)
{
     if (has_label(edge, confidential)) {
          add_label(node, confidential);
          if (node_type(node) == ENT_INODE_SOCKET)
               return PROV_RAISE_WARNING;
     }
     return 0;
}
```

```
static int out_edge(union prov_msg *node,
  union prov_msg *edge) {
     switch (edge_type(edge)) {
          case RL_WRITE:
          case RL_READ:
          case RL_SND:
          case RL_RCV:
          case RL_VERSION:
          case RL_VERSION_PROCESS:
          case RL_CLONE:
               if (has_label(node, confidential))
                    add_label(edge, confidential);
     }
     return 0;
}
```

```
QUERY_NAME(“Propagate labels”);
QUERY_DESCRIPTION(“Example query”);
QUERY_AUTHOR(“Not me.”);
QUERY_VERSION(“0.1”);
QUERY_LICENSE(“GPL”);
register_query(init, in_edge, out_edge);
```

CamQuery Implementation

Captures all information flow

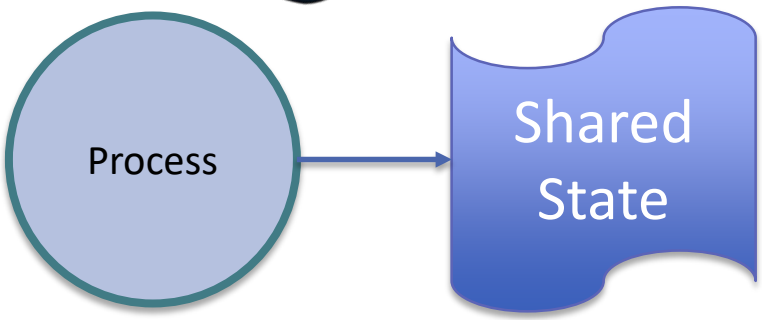


Process consumes Shared state

- CamFlow information flow patch. https://github. com/CamFlow/information-flow-patch.
- Laurent Georget, Mathieu Jaume, Guillaume Piolle, Frédéric Tronel, and Valérie Viet Triem Tong. 2017. Information Flow Tracking for Linux Handling Concurrent System Calls and Shared Memory. In International Conference on Software Engineering and Formal Methods. Springer, 1–16.
- Laurent Georget, Mathieu Jaume, Frédéric Tronel, Guillaume Piolle, and Valérie Viet Triem Tong. 2017. Verifying the reliability of operating system-level information flow control systems in Linux. In International Workshop on Formal Methods in Software Engineering (FormaliSE'17). IEEE/ACM, 10–16.

CamQuery Implementation

Captures all information flow

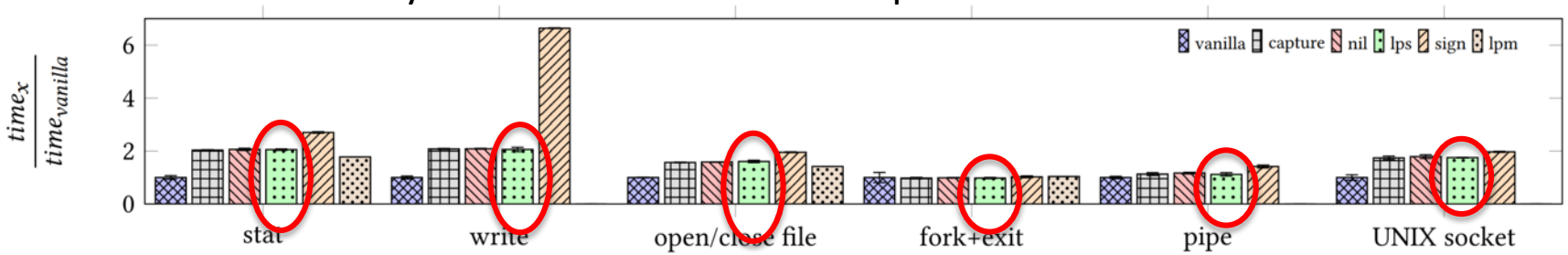


Process transmits to shared state

- CamFlow information flow patch. https://github. com/CamFlow/information-flow-patch.
- Laurent Georget, Mathieu Jaume, Guillaume Piolle, Frédéric Tronel, and Valérie Viet Triem Tong. 2017. Information Flow Tracking for Linux Handling Concurrent System Calls and Shared Memory. In International Conference on Software Engineering and Formal Methods. Springer, 1–16.
- Laurent Georget, Mathieu Jaume, Frédéric Tronel, Guillaume Piolle, and Valérie Viet Triem Tong. 2017. Verifying the reliability of operating system-level information flow control systems in Linux. In International Workshop on Formal Methods in Software Engineering (FormaliSE'17). IEEE/ACM, 10–16.

Performance

Syscall slowdown relative to plain Linux Kernel

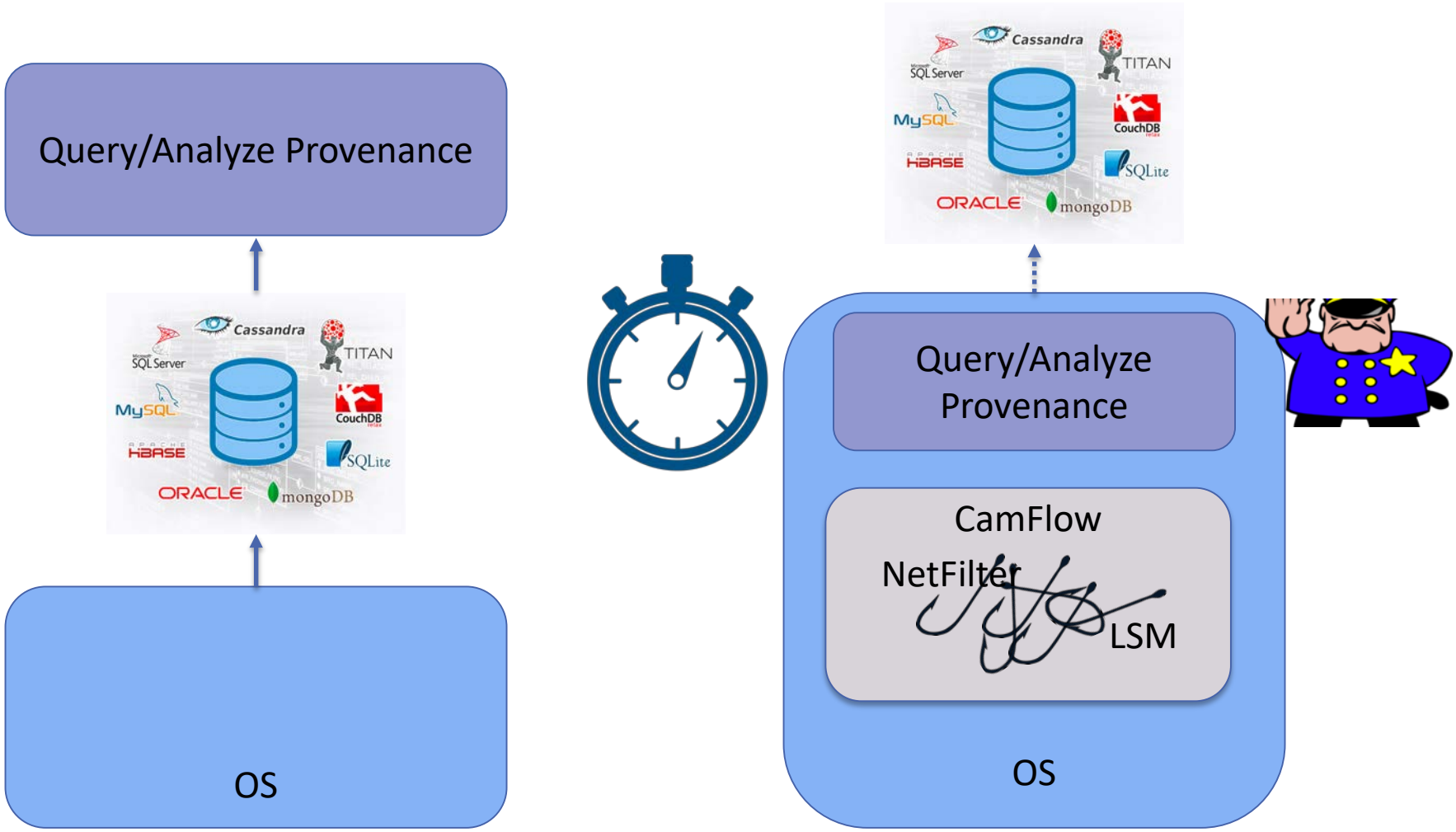


Macrobenchmark Performance

Test Type	vanilla	capture	nil	lps	sign	PASS	LPM
Execution time in seconds, smaller is better							
unpack	14.98	15.48 (3%)	15.63 (4%)	15.76 (5%)	16.68 (11%)	NA	NA
build	402	411 (2%)	416 (3%)	417 (3%)	448 (11%)	15.6%	2.7%
4kB to 1MB file, 10 subdirectories, 4k5 simultaneous transactions, 1M5 transactions							
postmark	127	145 (14%)	144 (13%)	146 (15%)	226 (78%)	11.5%	7.5%

Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. 2018. Runtime Analysis of Whole-System Provenance. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). ACM, New York, NY, USA, 1601-1616. DOI: https://doi.org/10.1145/3243734.3243776

CamQuery Wrap Up



Fun with Non-Volatile Memory

Adapting Existing Solutions

Keys	Values
Session1	Cidon, Manno, Evans, Guyot
Session2	Blomer, Hallak, Bbrown, Manno
Session3	Strauss, Peglar, Gervasi
Sesson4	Lightning Talks

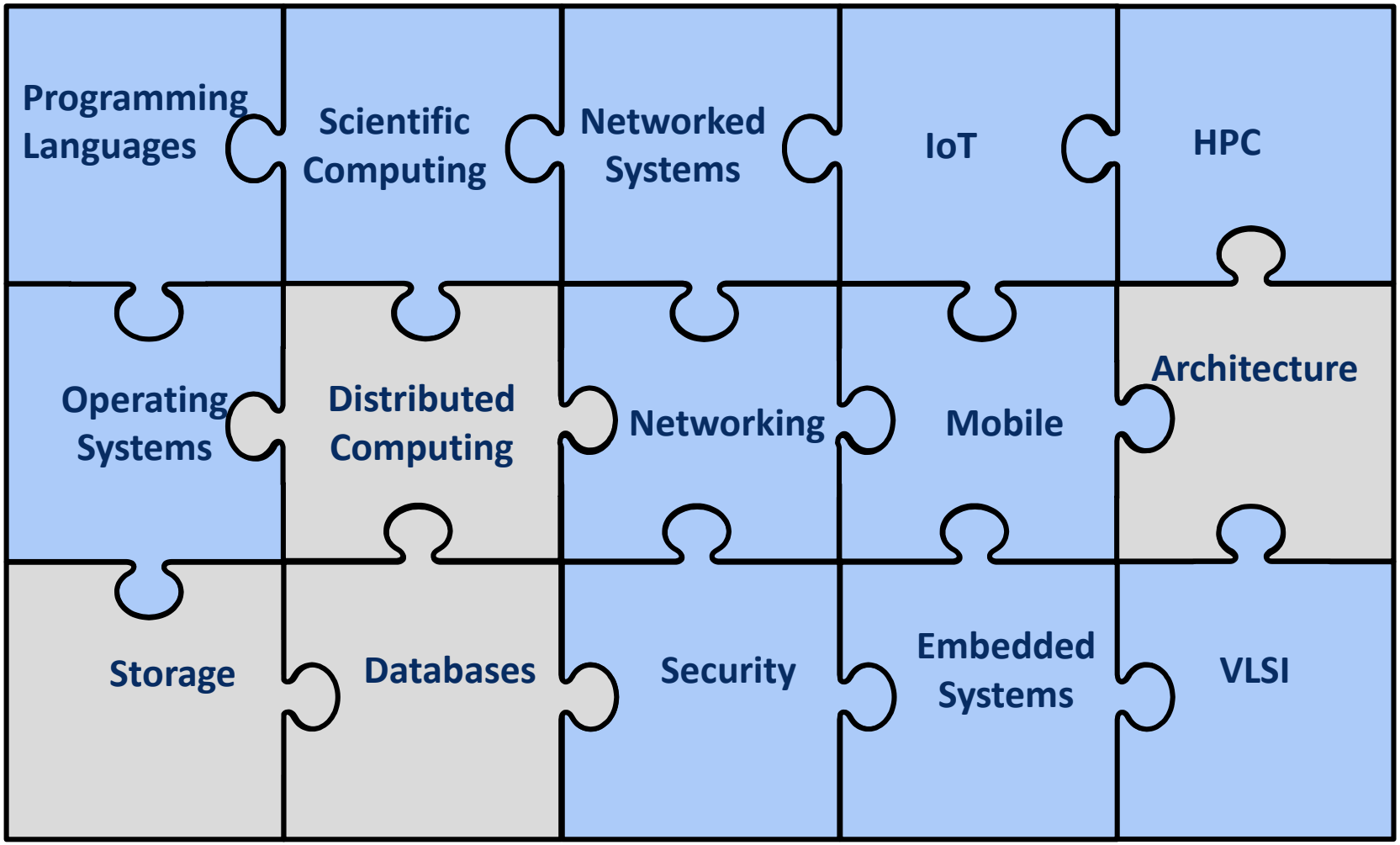
Building Custom Solutions

Keys	Values
Session1	Cidon, Manno, Evans, Guyot
Session2	Blomer, Hallak, Bbrown, Manno
Session3	Strauss, Peglar, Gervasi
Sesson4	Lightning Talks

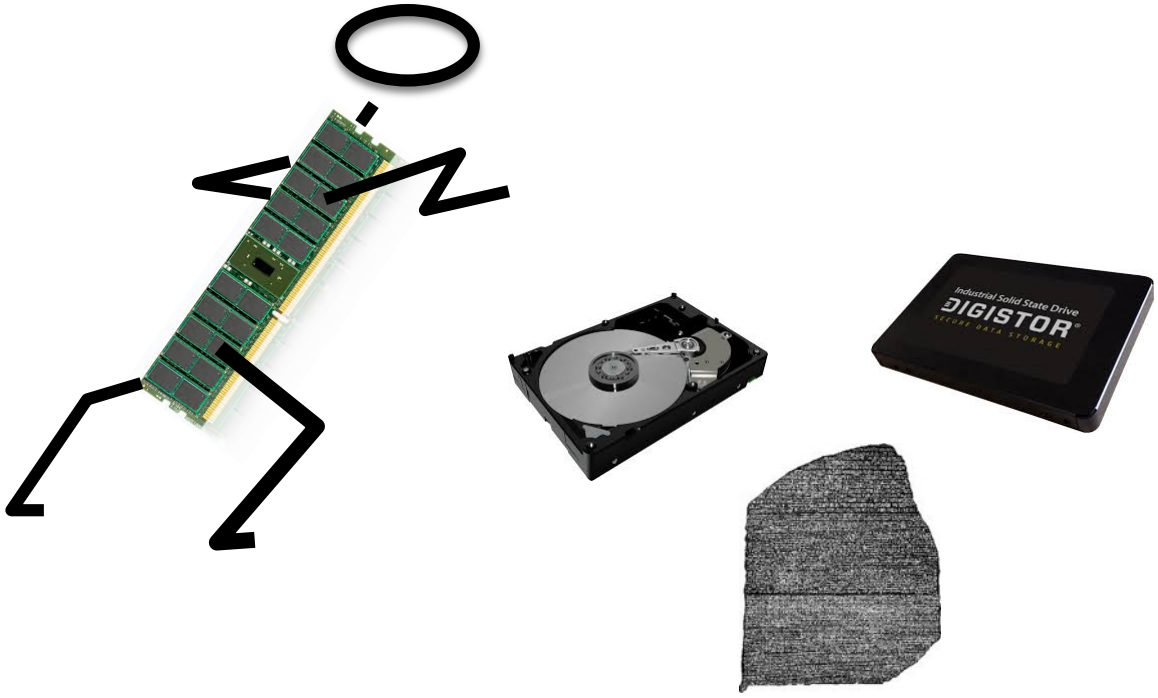
Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

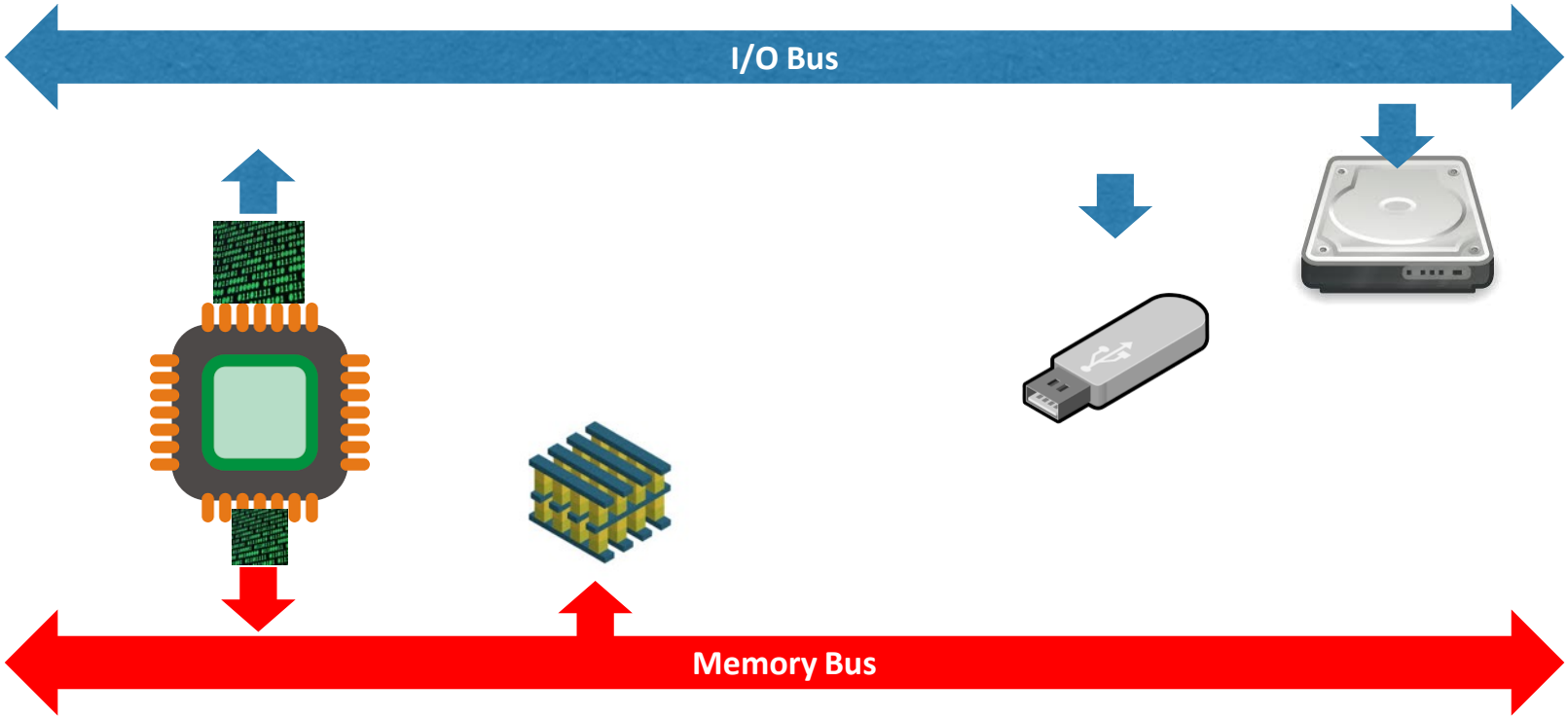
Fun with Non-Volatile Memory



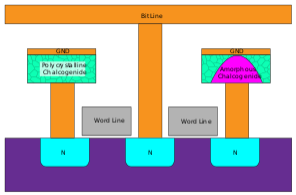
NVM Characteristics



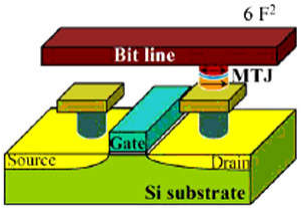
NVM 101



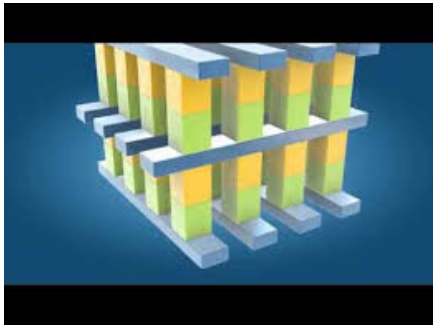
3D Xpoint

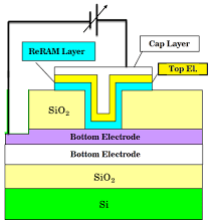


PCM

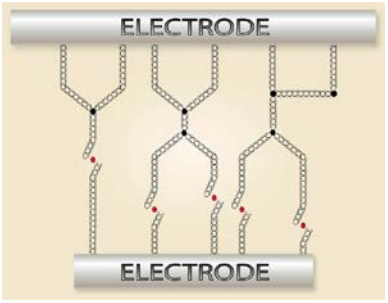


STT-RAM





ReRAM



Carbon Nanotubes

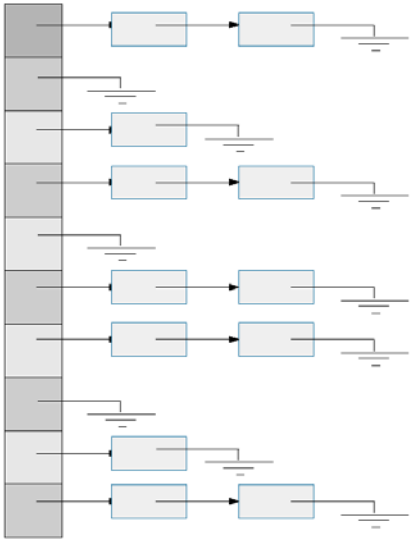
https://www.embedded.com/design/real-time-and-performance/4026000/The-future-of-scalable-STT-RAM-as-a-universal-embedded-memory
https://en.wikipedia.org/wiki/Phase-change_memory
http://nantero.com/technology/

What Happens When you try to make an in-memory KV Store persistent?

1. Data structure modification is contagious
2. Failure-atomic transactions are crucial
3. Persistent and nonpersistent objects interact in unexpected ways
4. Concurrency is hard

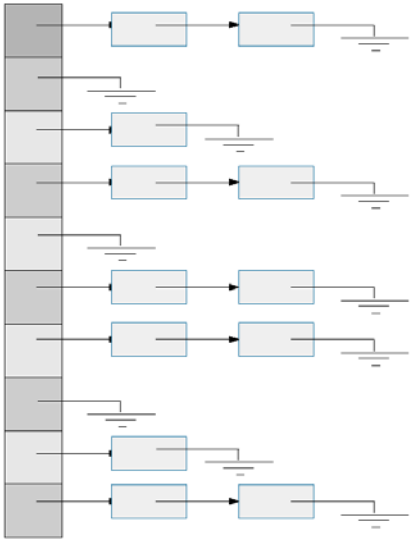
Persistent Memcached: Bringing Legacy Code to Byte-Addressable Persistent Memory (PDF)
Marathe, V., Seltzer, M., Byan, S., Harris, T.
Proceedings of USENIX Workshop on Hot Topics in Storage and File Systems 2017

Lesson I: Modifications are contagious



Central Hash Table

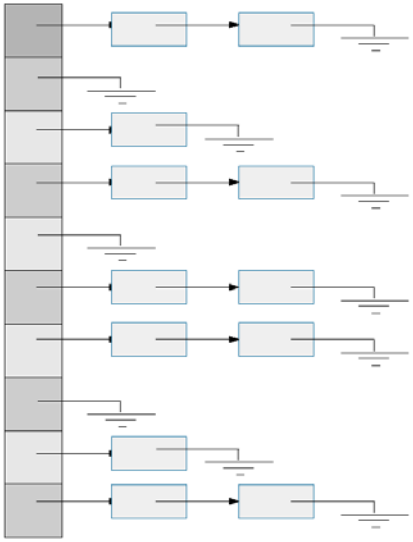
Lesson I: Modifications are contagious



Central Hash Table

LRU Cache

Lesson I: Modifications are contagious

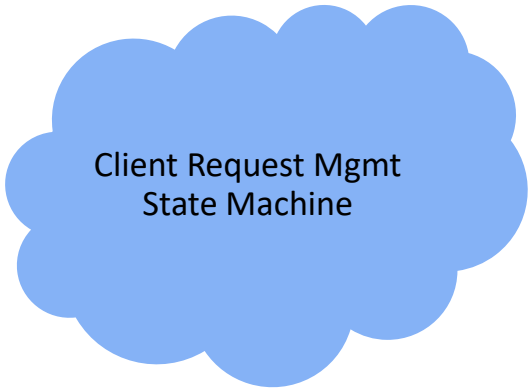


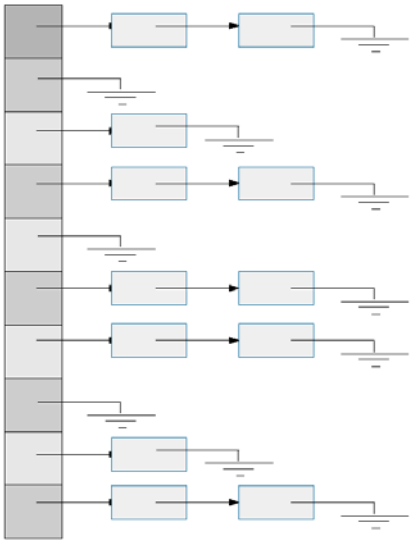
Central Hash Table

LRU Cache

Slab Allocator

Lesson I: Modifications are contagious



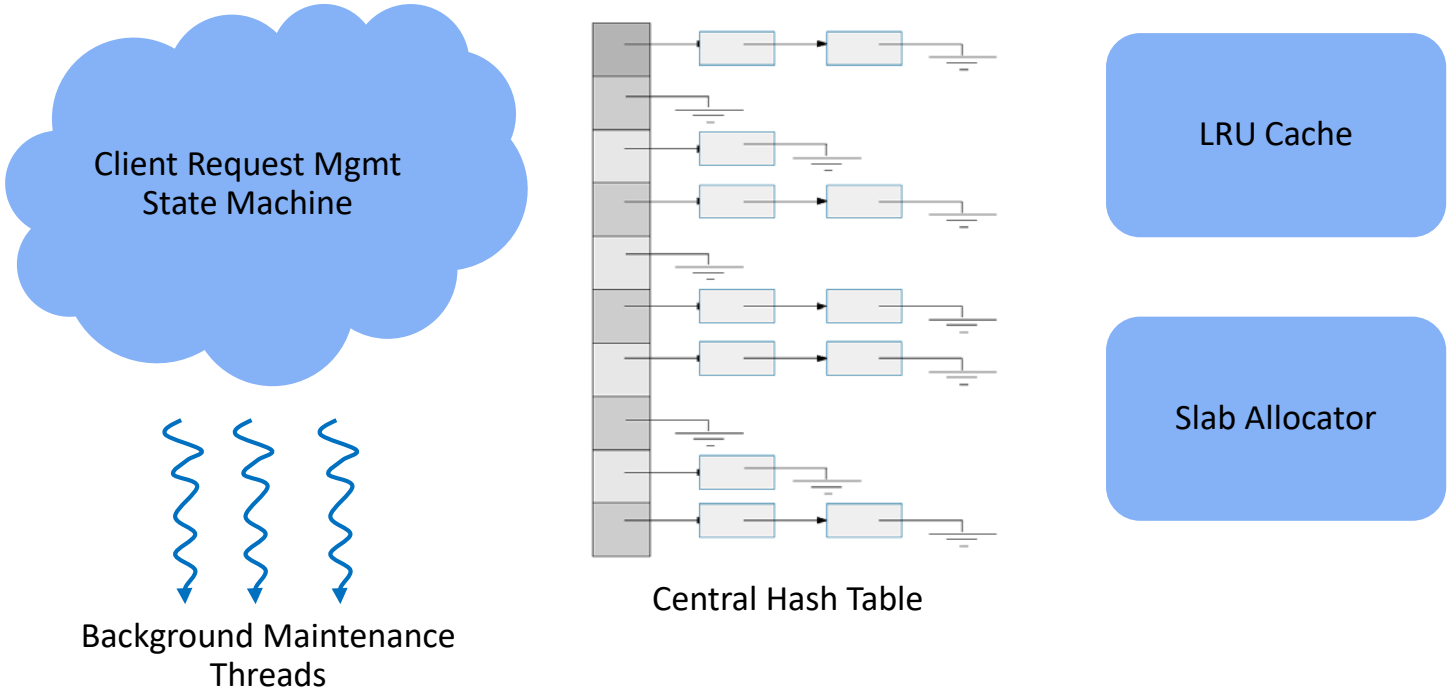


Central Hash Table

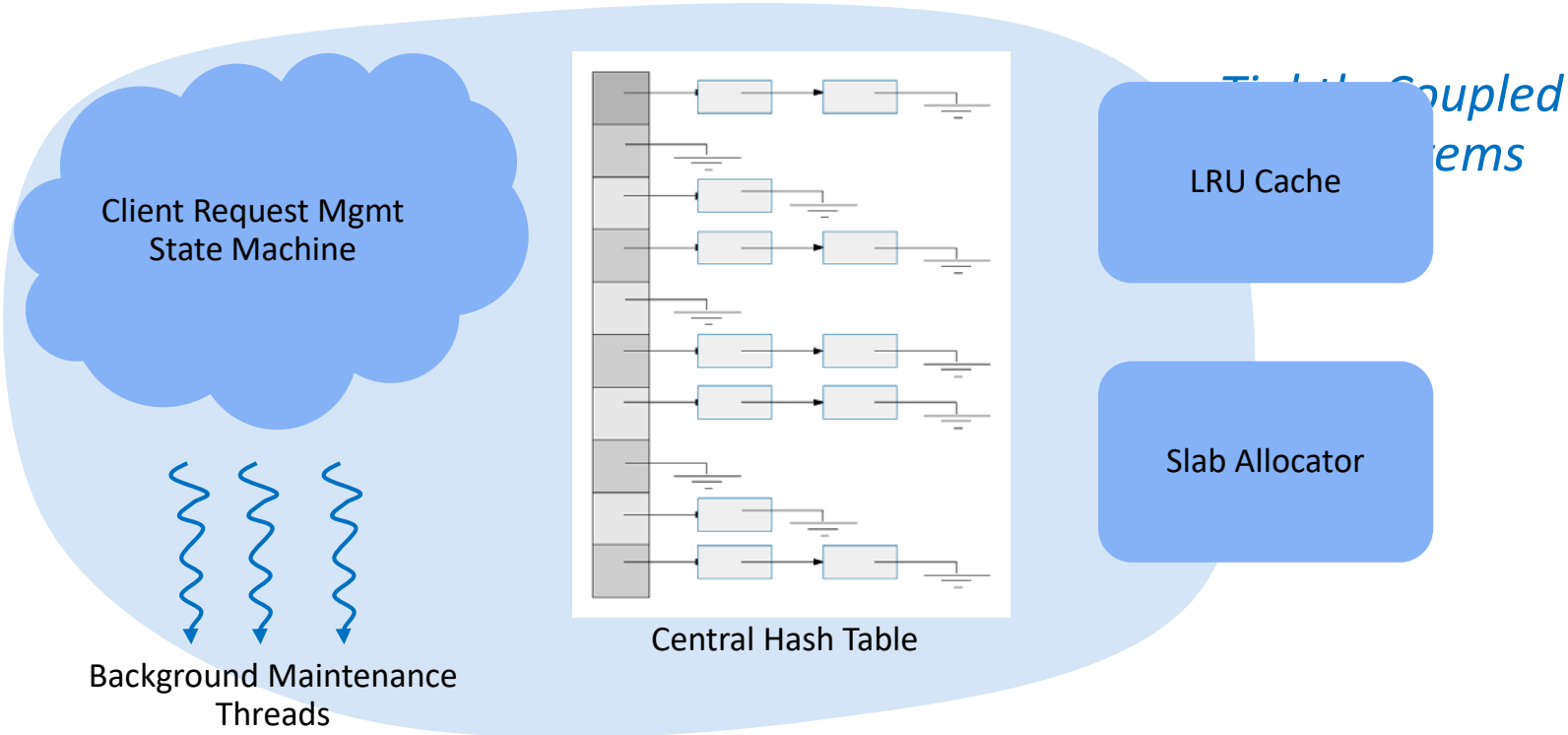
LRU Cache

Slab Allocator

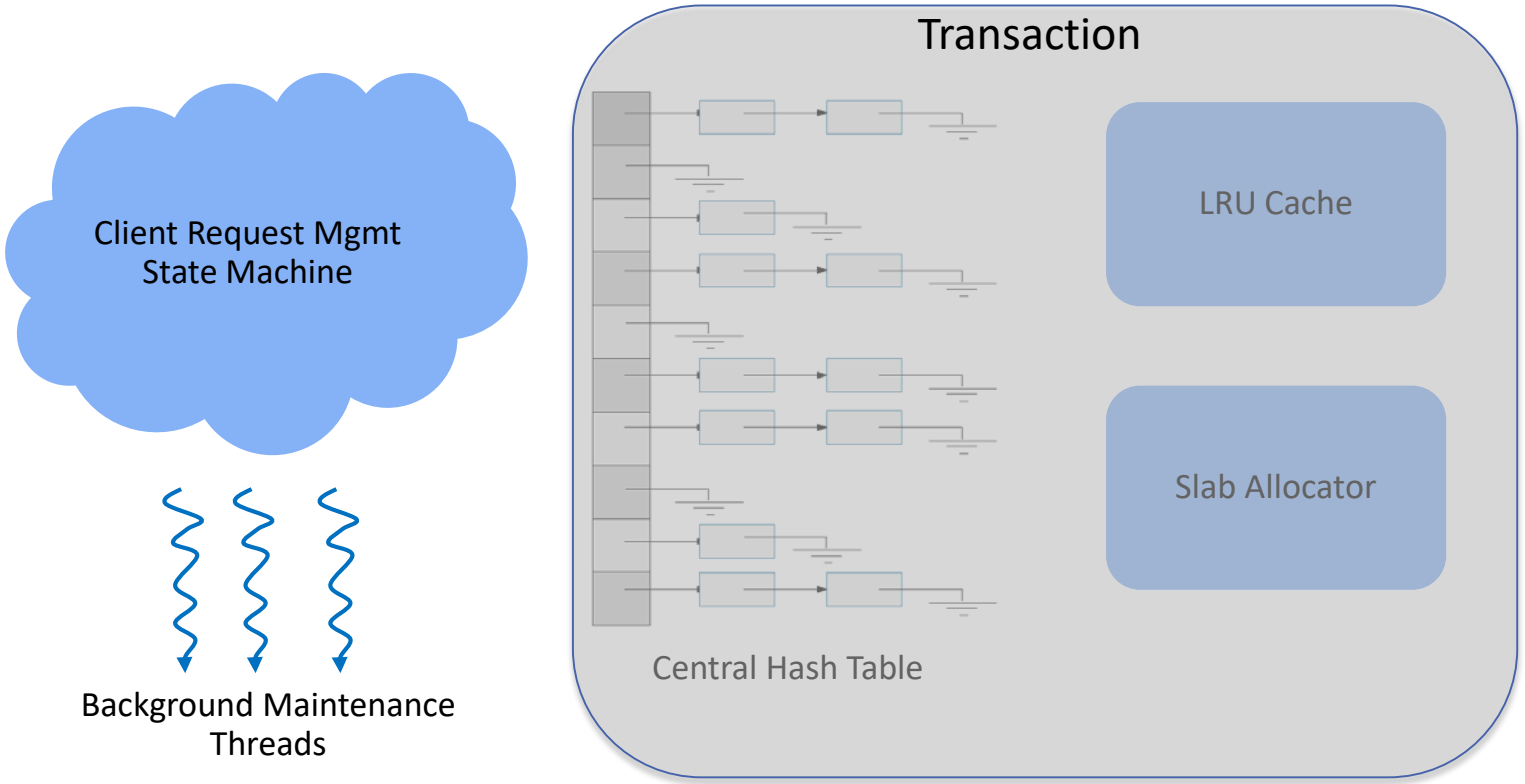
Lesson I: Modifications are contagious



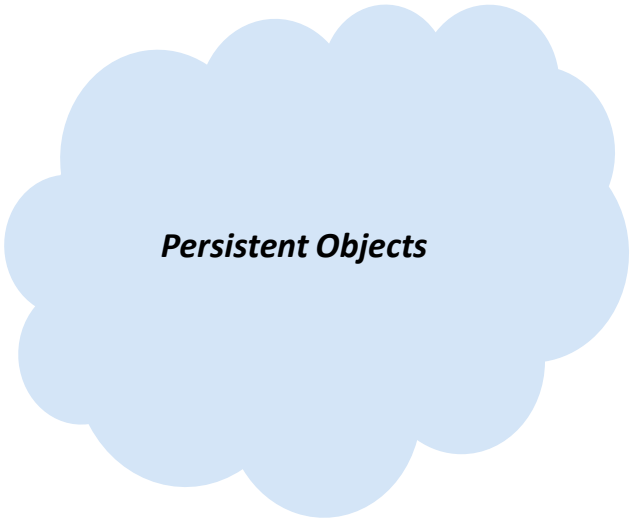
Lesson I: Modifications are contagious



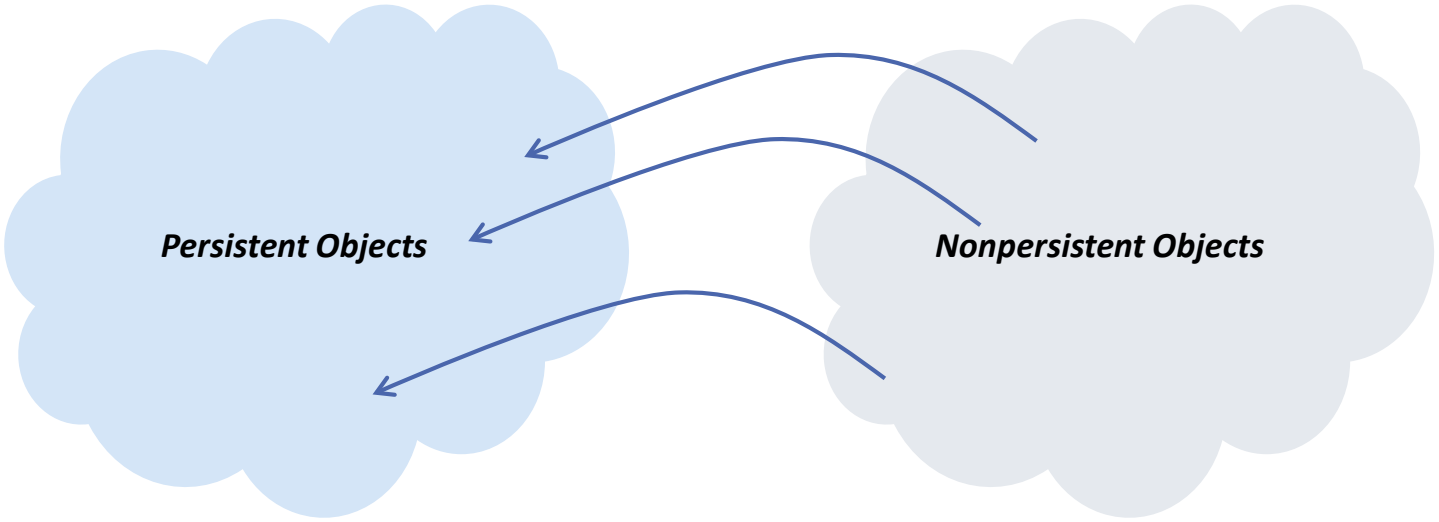
Lesson II: Transactions!



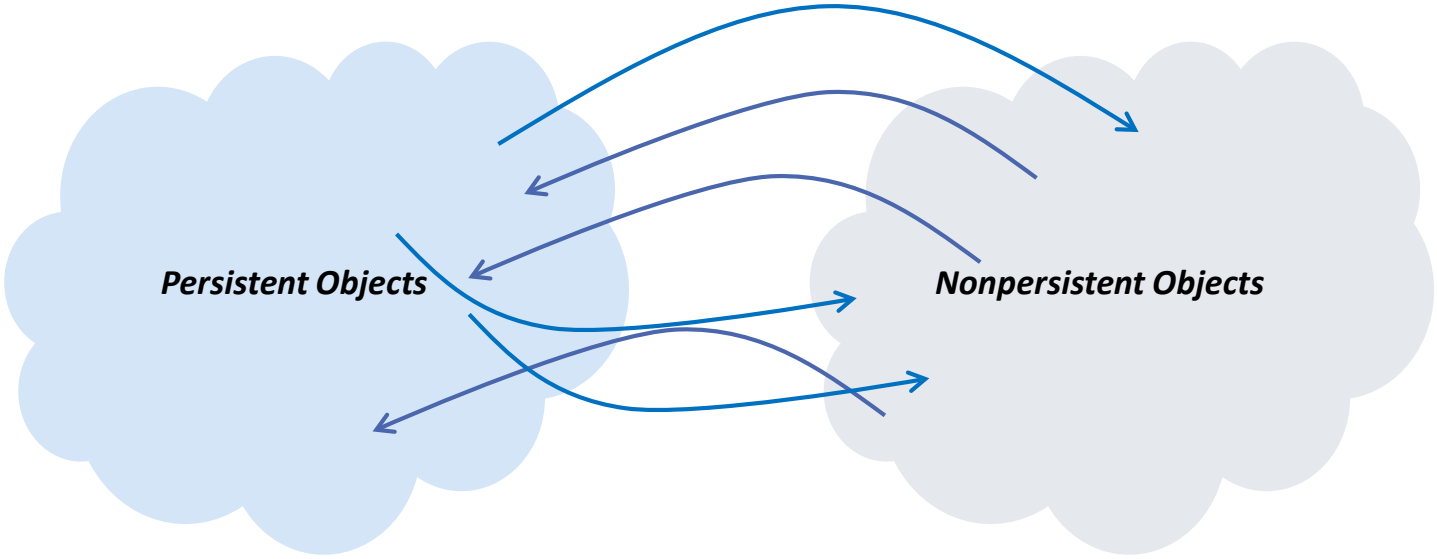
Lesson III: Persistent and nonpersistent objects interact in unexpected ways



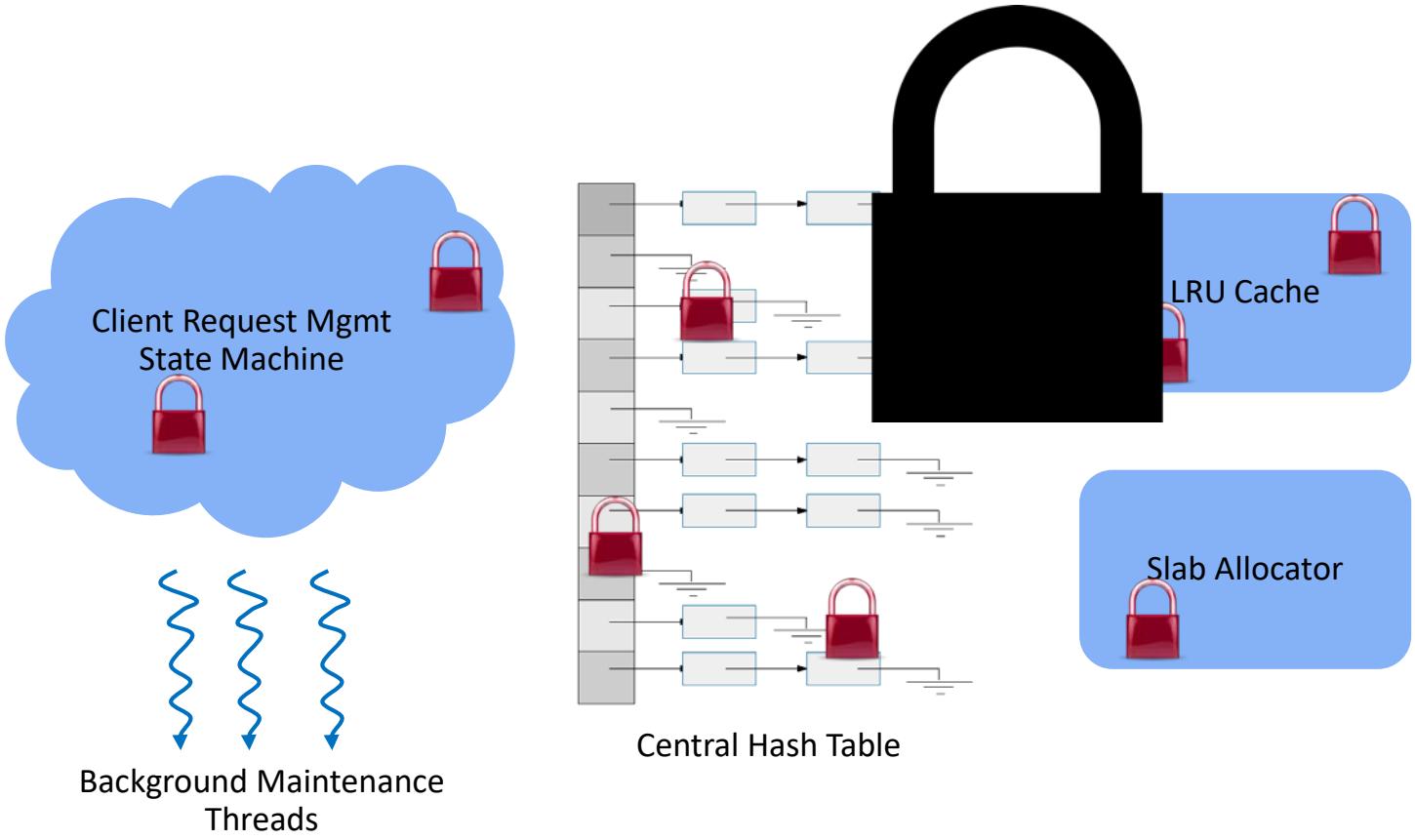
Lesson III: Persistent and nonpersistent objects interact in unexpected ways



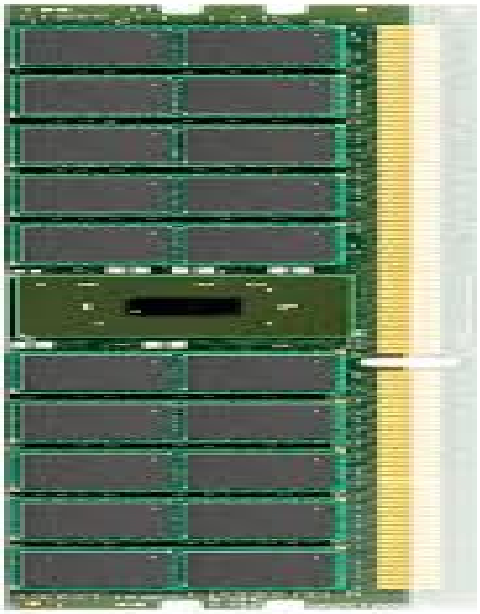
Lesson III: Persistent and nonpersistent objects interact in unexpected ways

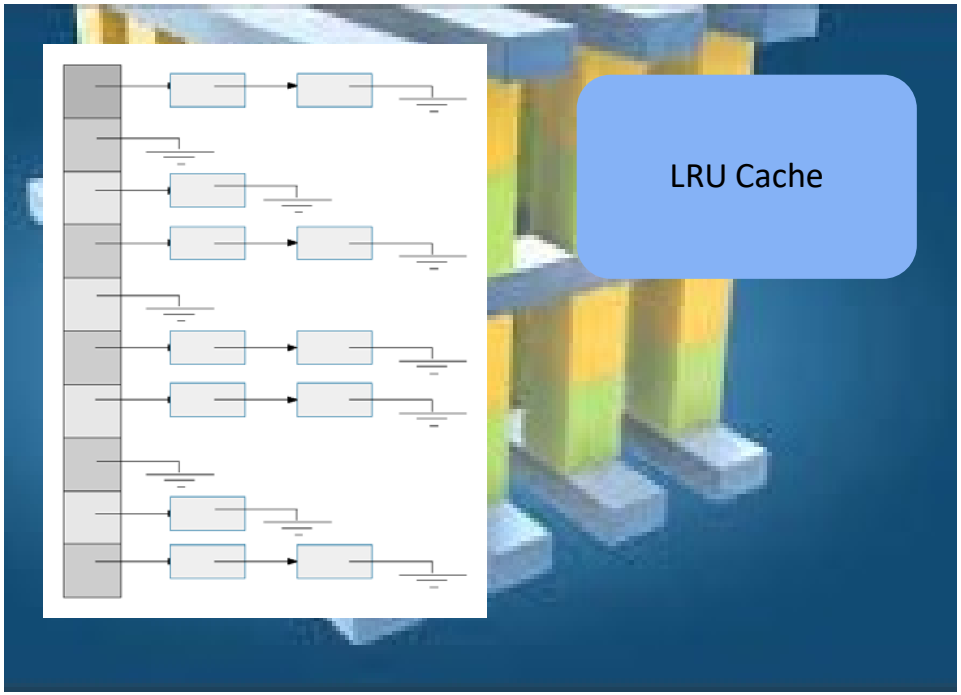


Lesson IV: Concurrency is Hard



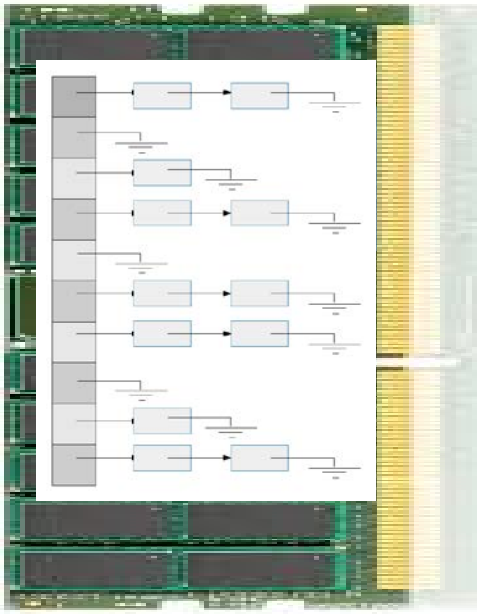
BULLET: A Hybrid DRAM/NVM Hash Table





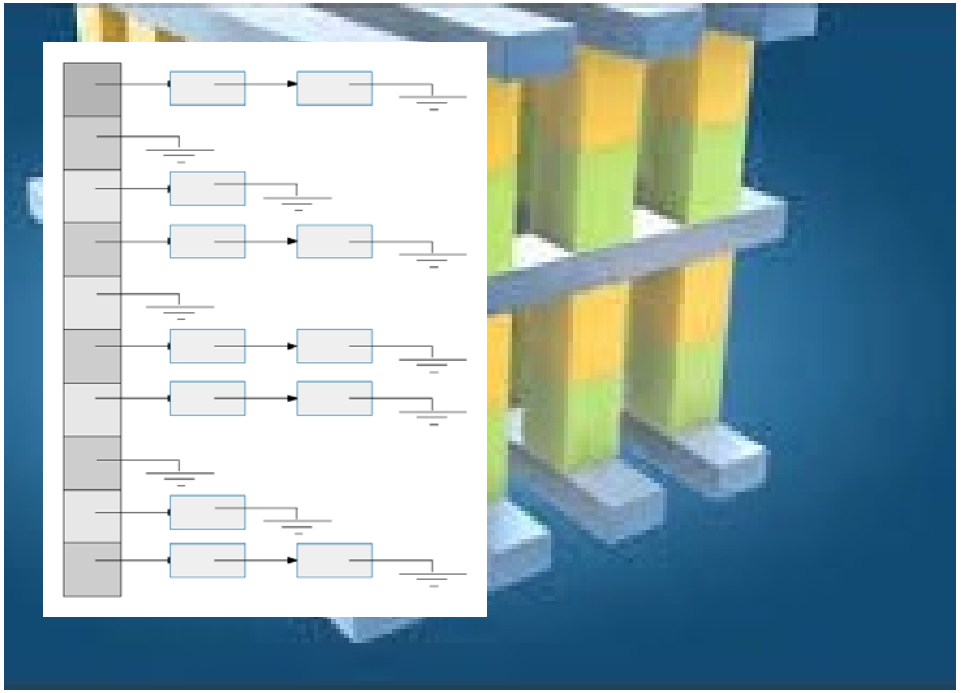
Closing the Performance Gap Between Volatile and Persistent Key-Value Stores Using Cross-Referencing Logs
Huang, Y., Pavlovic, M., Marathe, V., Seltzer, M., Harris, T., Byan, S.
Proceedings of the 2018 USENIX Annual Technical Conference, Boston MA, June 2018.

BULLET: A Hybrid DRAM/NVM Hash Table

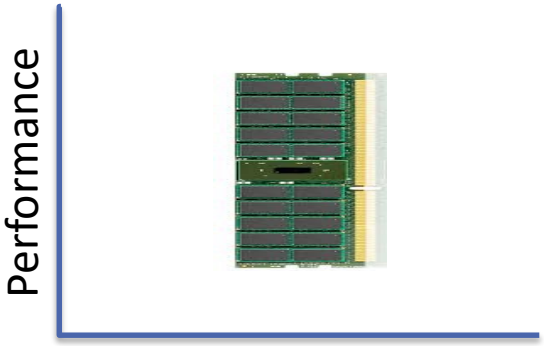


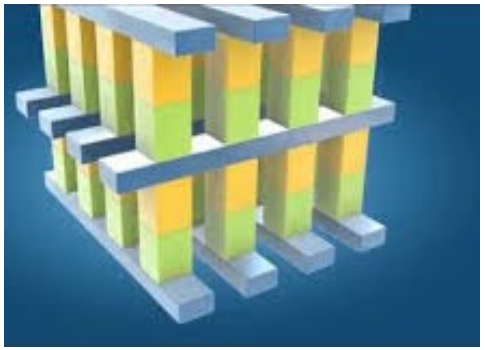
Front End

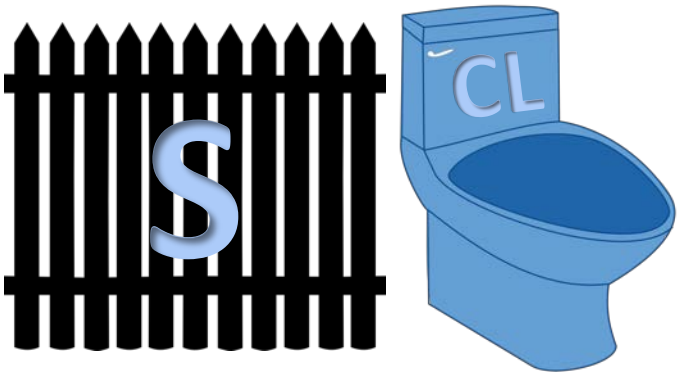
Back End



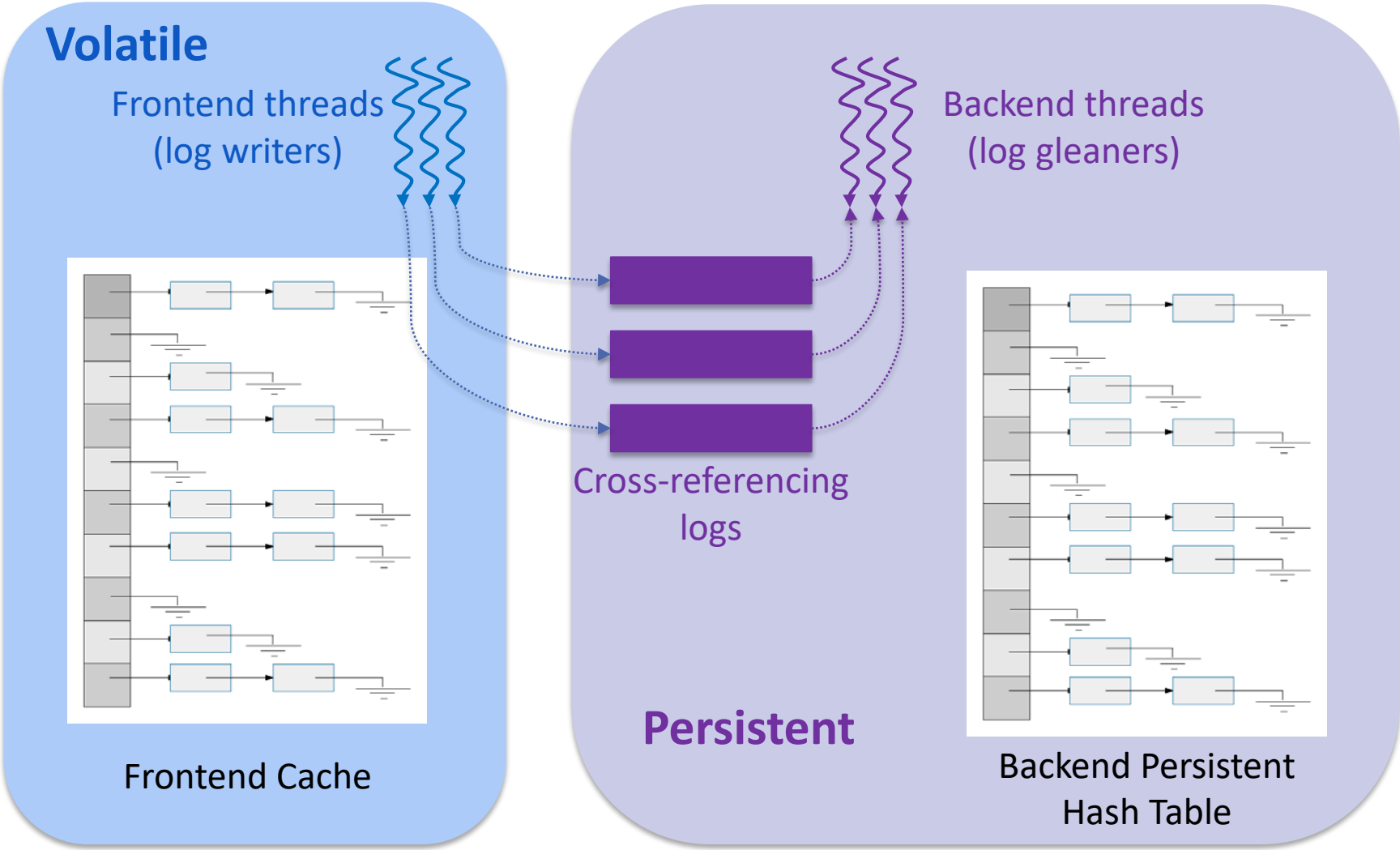
Challenges



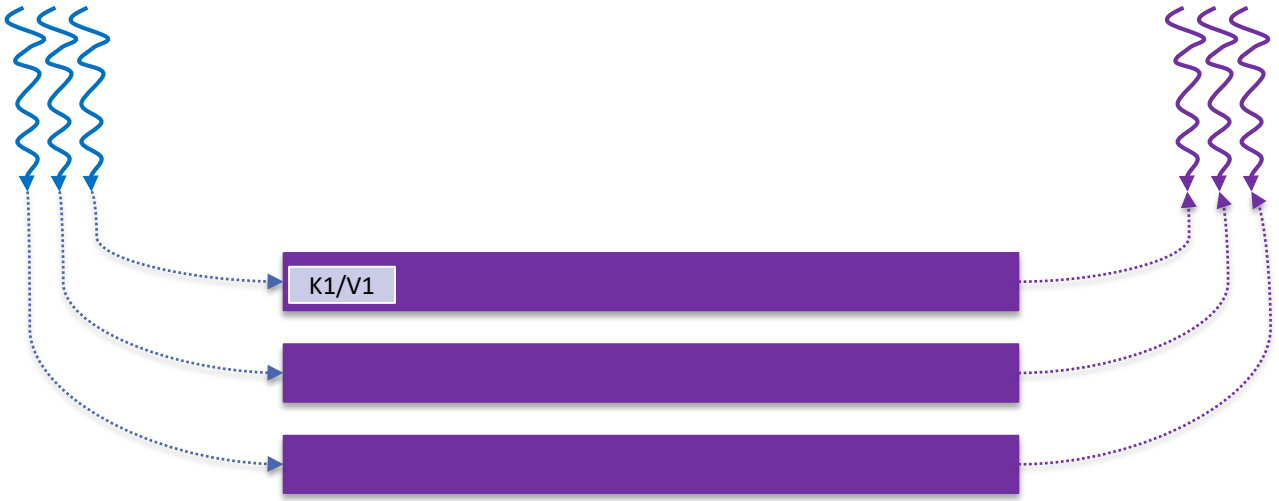




Bullet Architecture



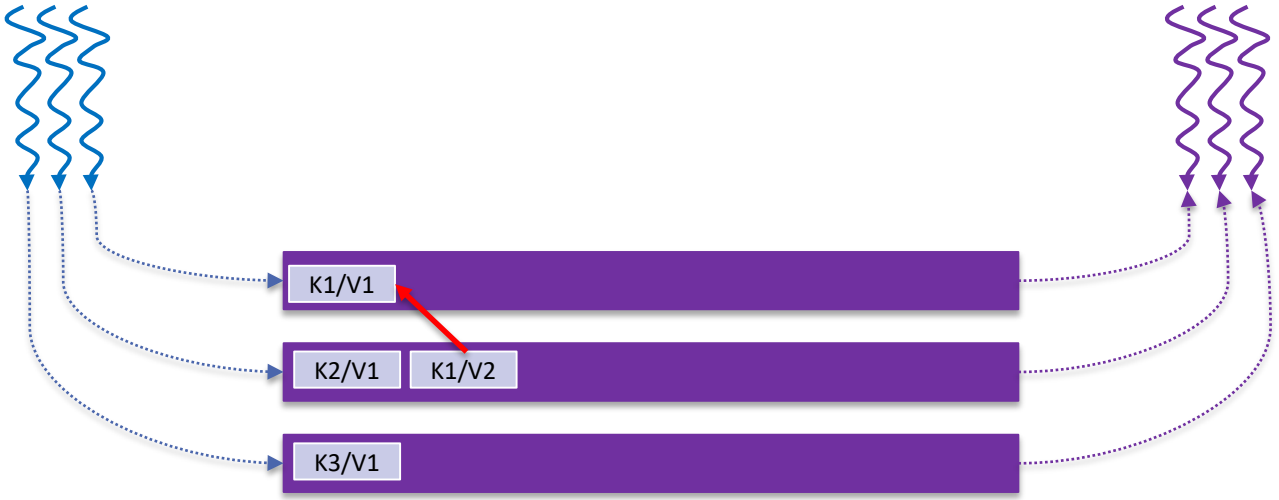
Cross-Referencing Logs



Log Record

len	klen	opcode	applied	epoch	prev	Key/value
26	2	append	NO	1	NULL	K1/V1

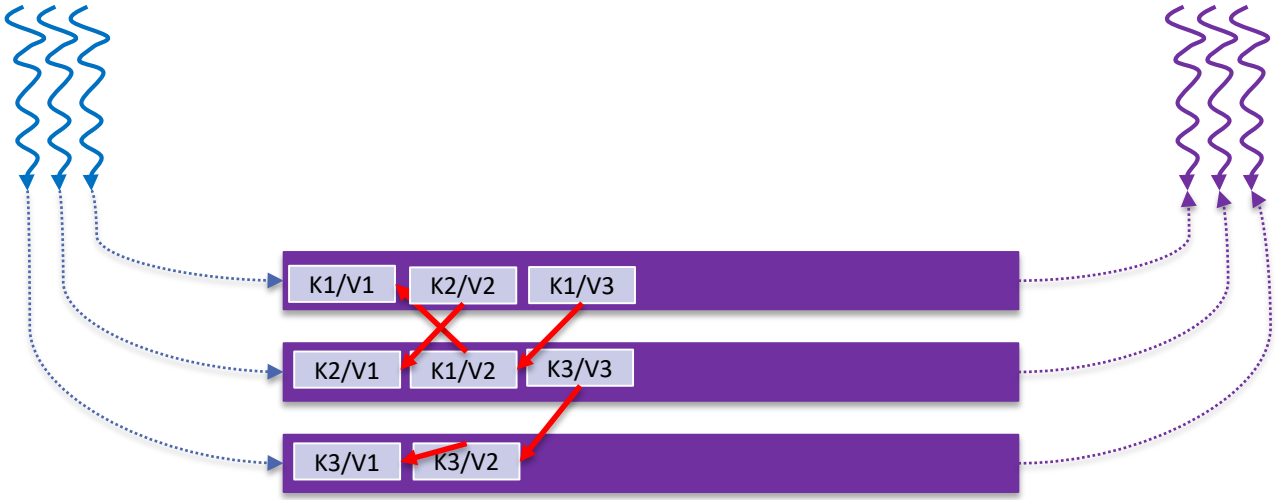
Cross-Referencing Logs



Log Record

len	klen	opcode	applied	epoch	prev	Key/value

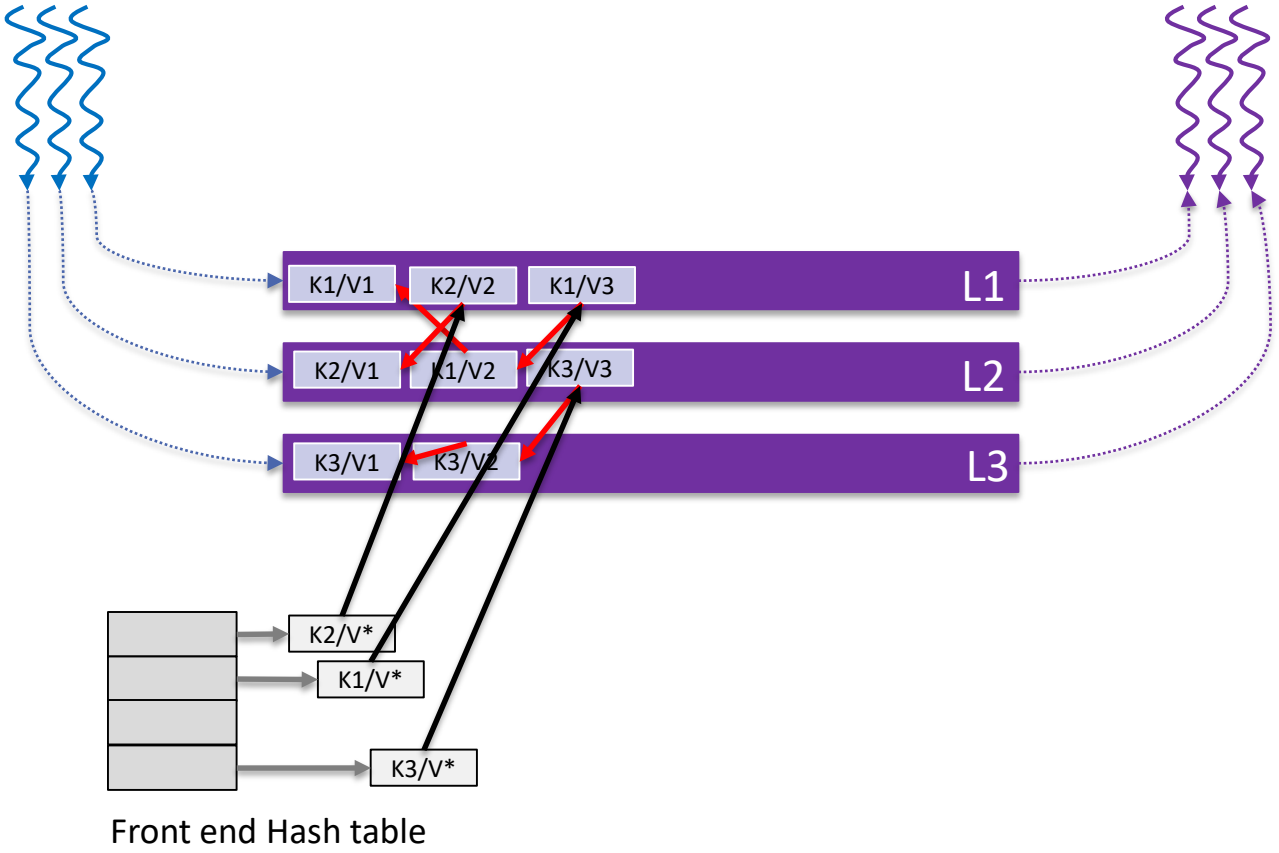
Cross-Referencing Logs



Log Record

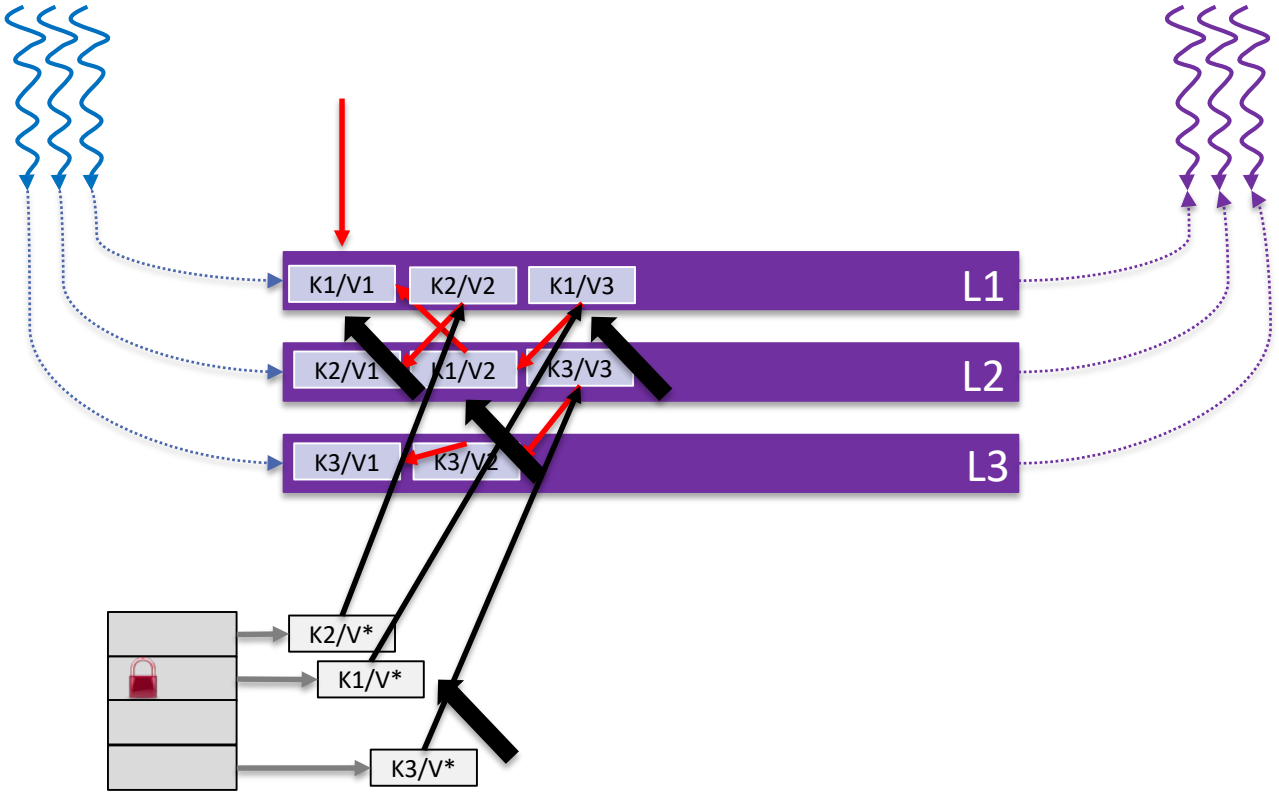
len	klen	opcode	applied	epoch	prev	Key/value

Cross-Referencing Logs



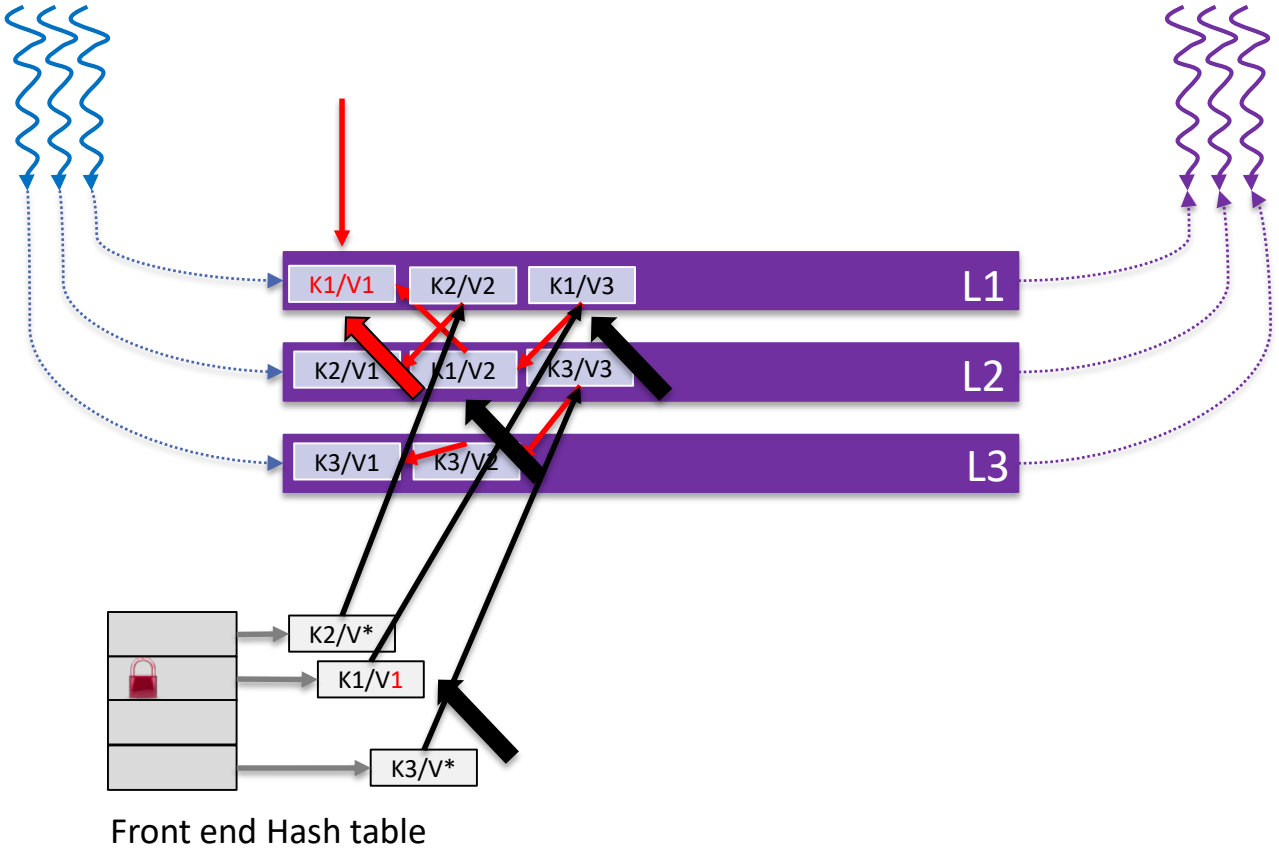
Front end Hash table

Applying Log Records



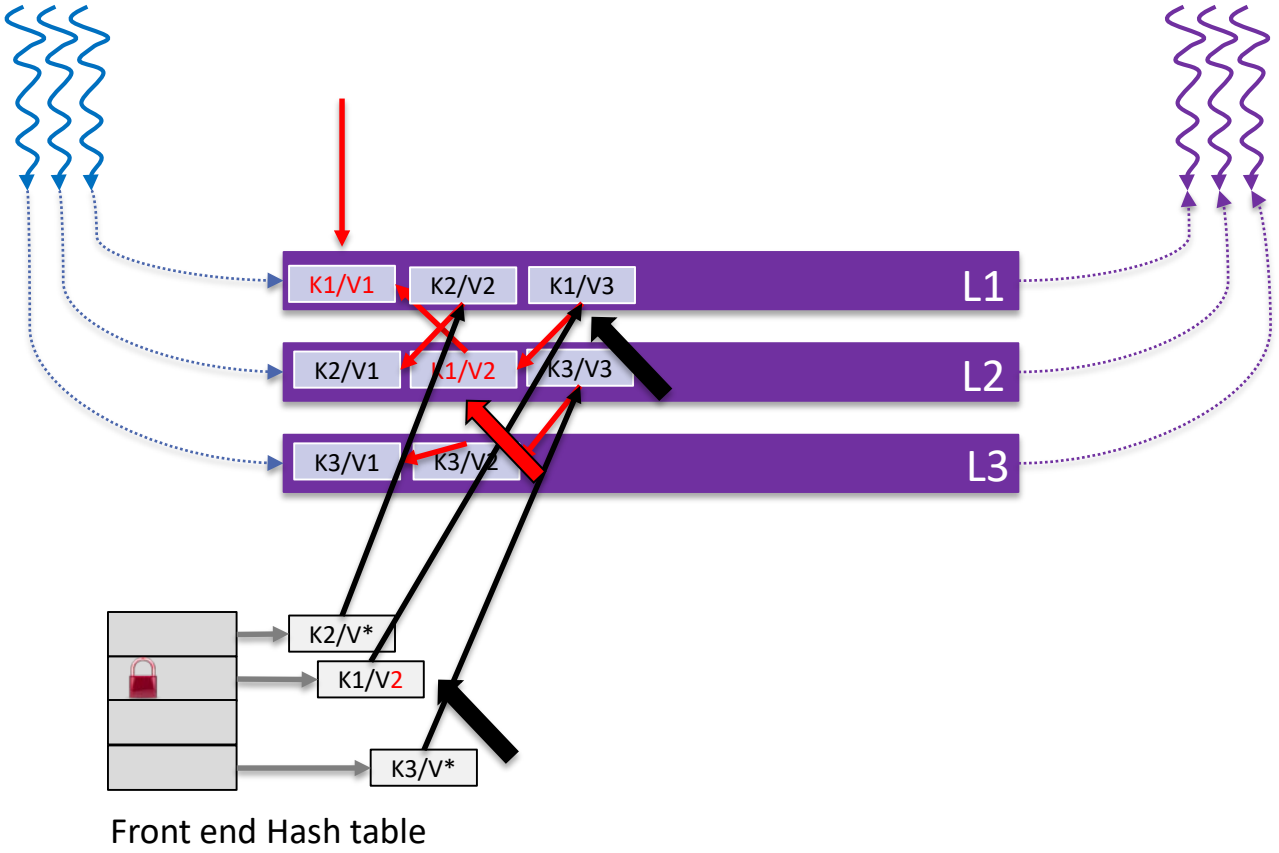
Front end Hash table

Applying Log Records



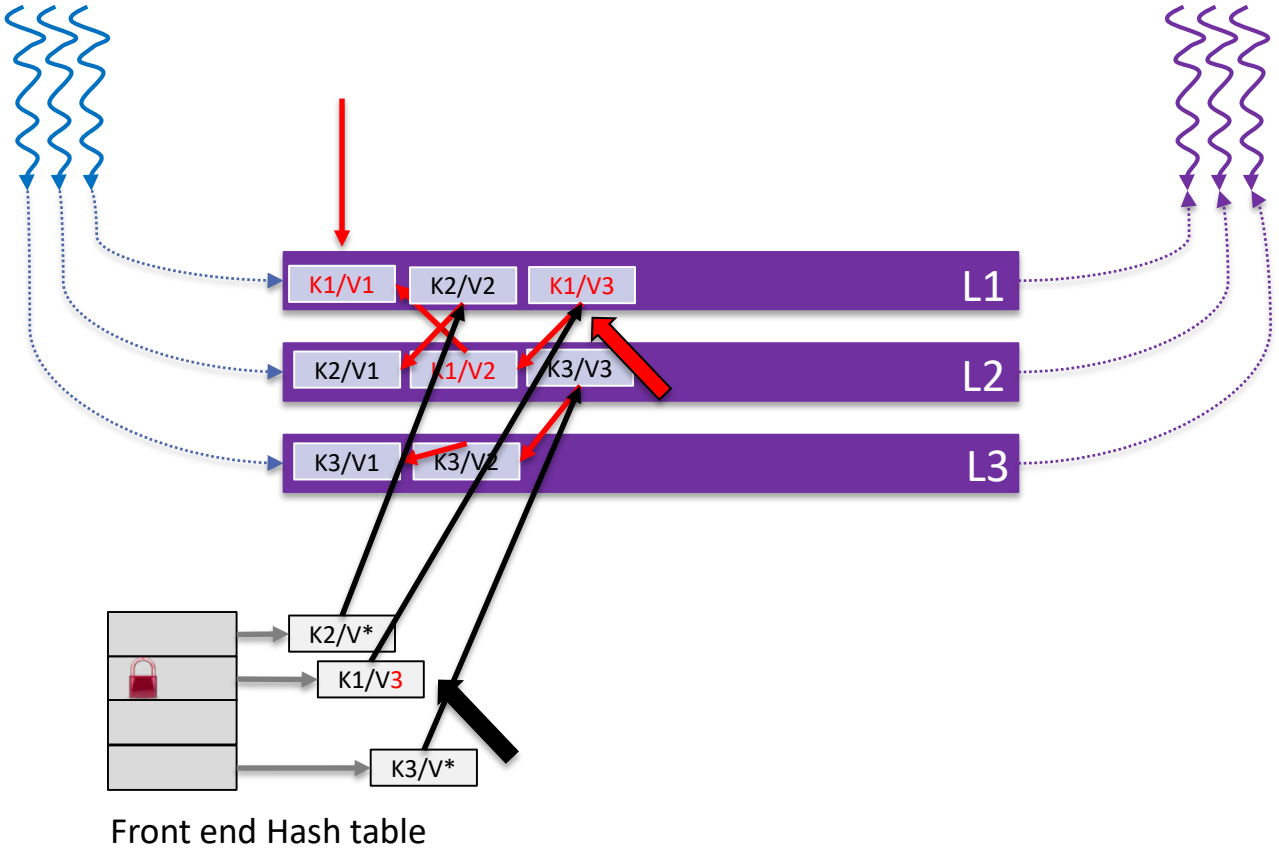
Front end Hash table

Applying Log Records

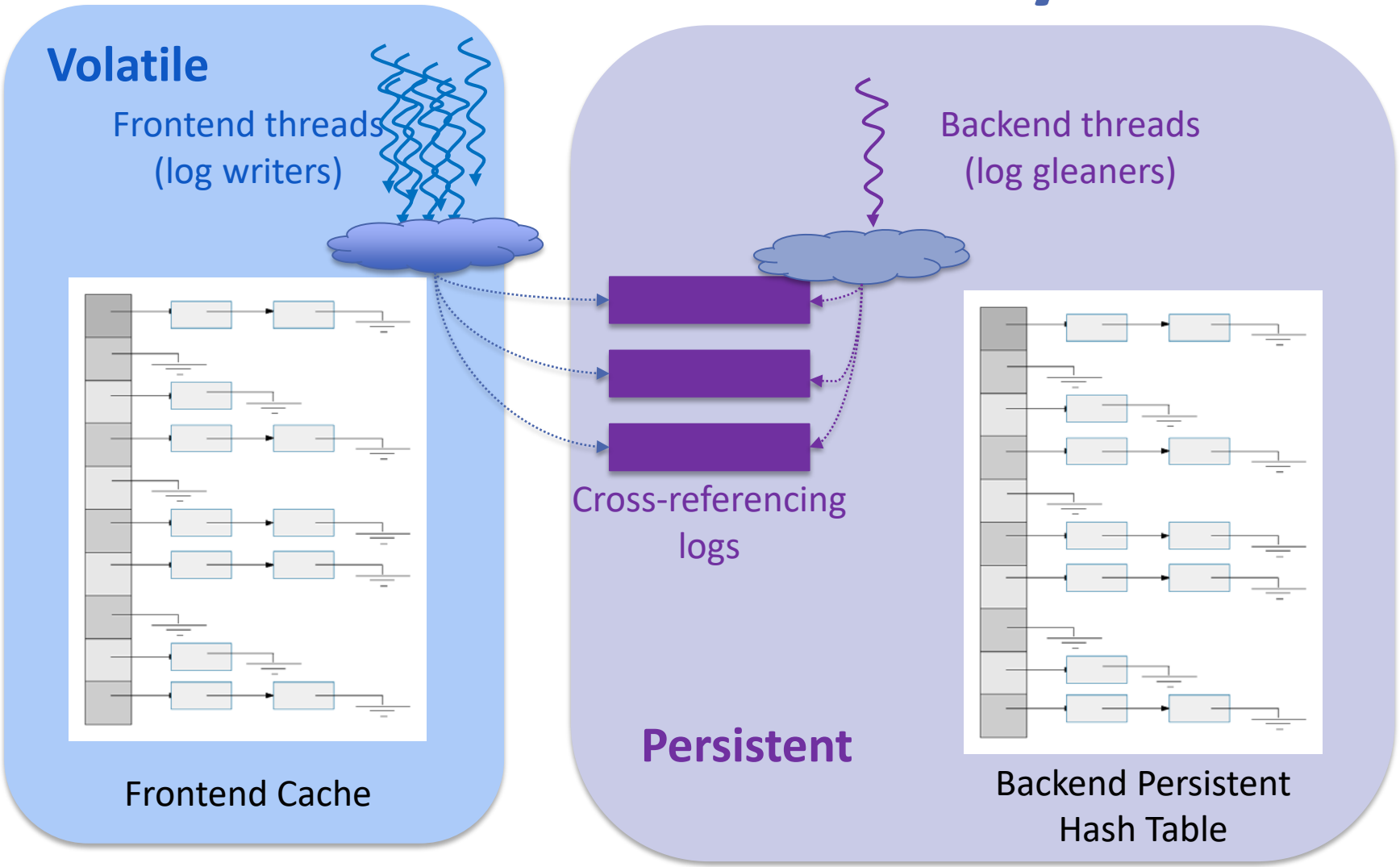


Front end Hash table

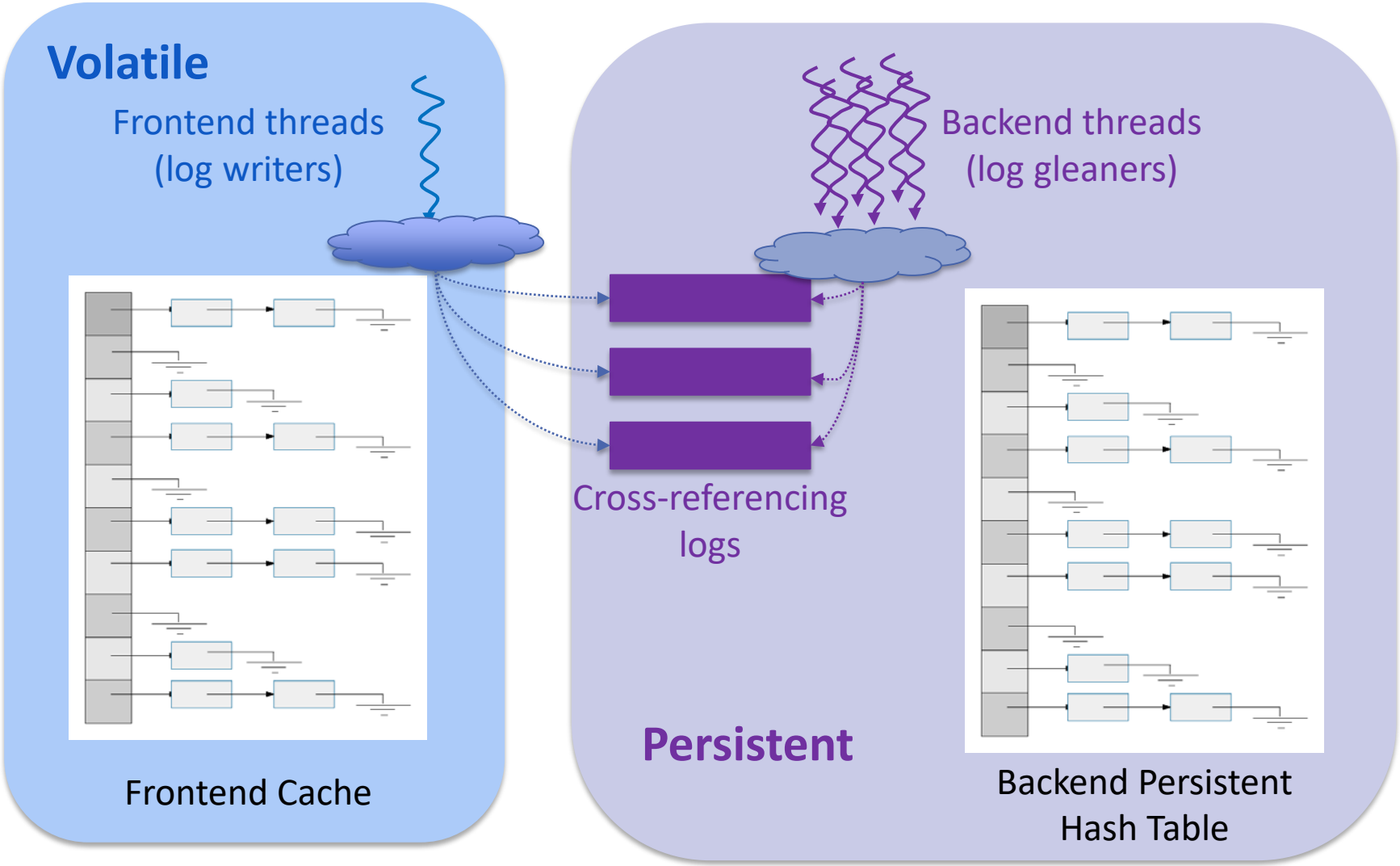
Applying Log Records



Bullet: Read Heavy



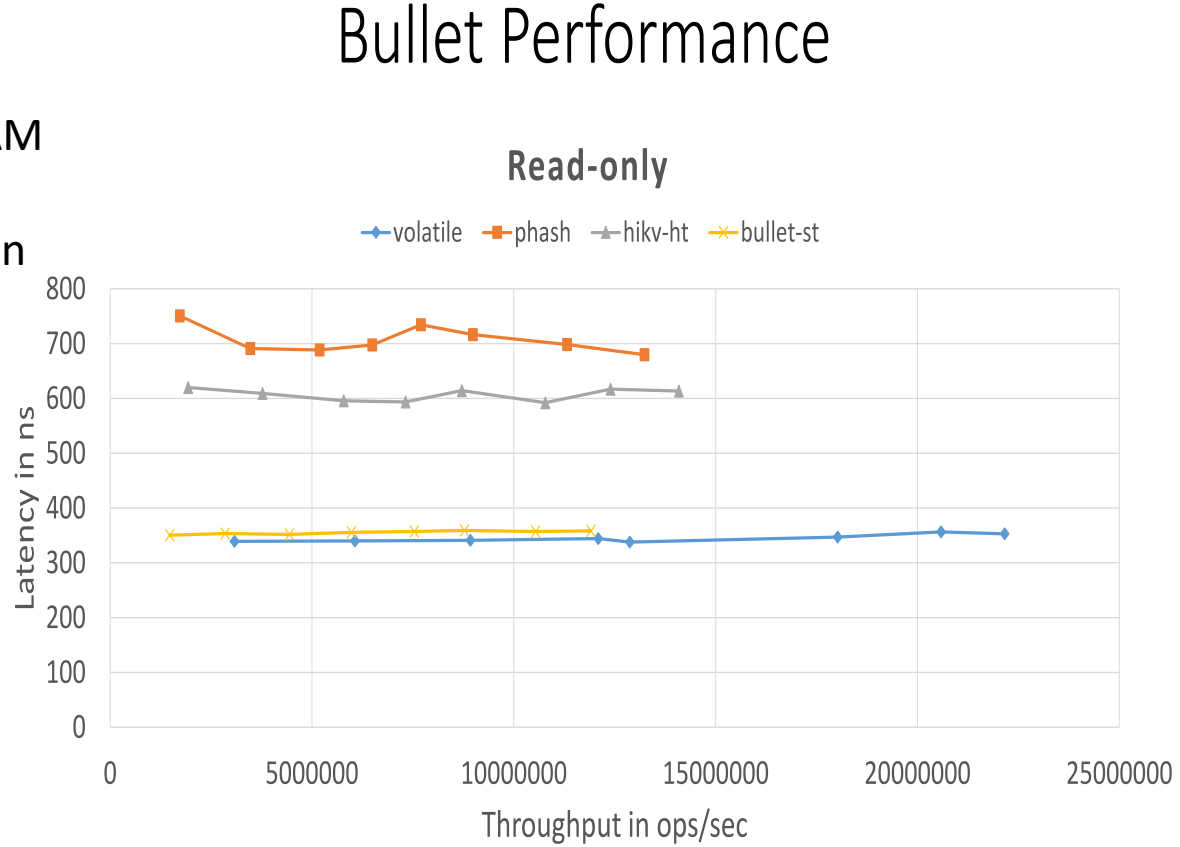
Bullet: Write Heavy



How Does it Perform?

Experimental Setup

- 16 cores; 512 GB DRAM
- Intel's NVM emulator
- Zipfian key distribution
- Show 99%-ile latency
- Comparison **HiKV**



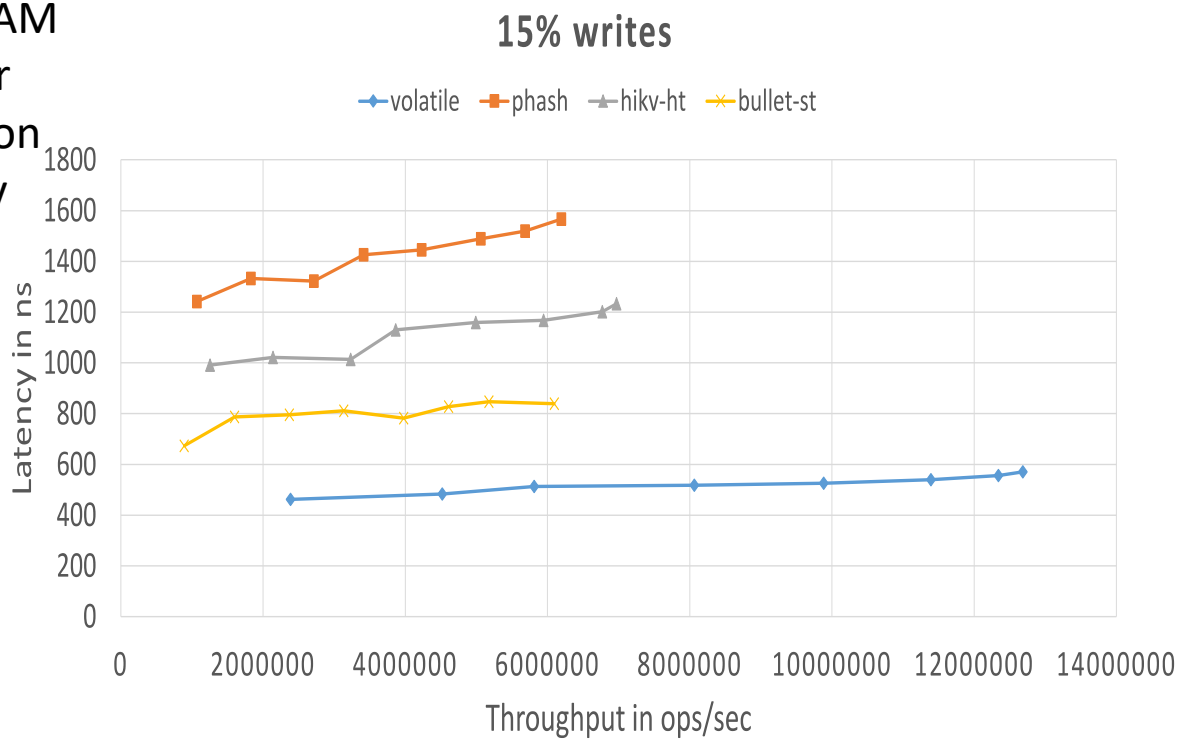
HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems
Fei Xia, *Institute of Computing Technology, Chinese Academy of Sciences; University of Chinese Academy of Sciences;* Dejun Jiang, Jin Xiong, and Ninghui Sun, *Institute of Computing Technology, Chinese Academy of Sciences*
Proceedings of the 2017 USENIX Annual Technical Conference, Santa Clara CA, July 2017.

How Does it Perform?

Bullet Performance

Experimental Setup

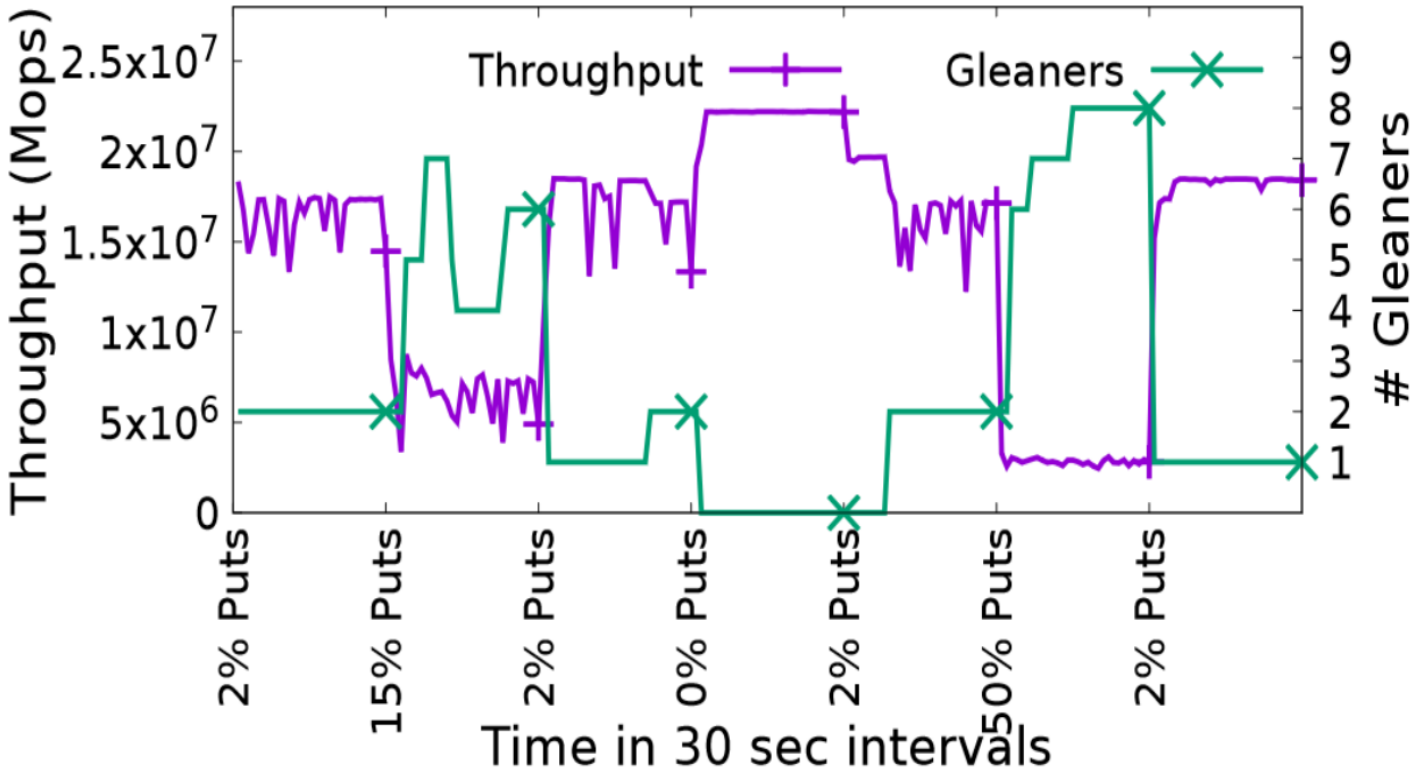
- 16 cores; 512 GB DRAM
- Intel's NVM emulator
- Zipfian key distribution
- Show 99%-ile latency
- Comparison **HiKV**



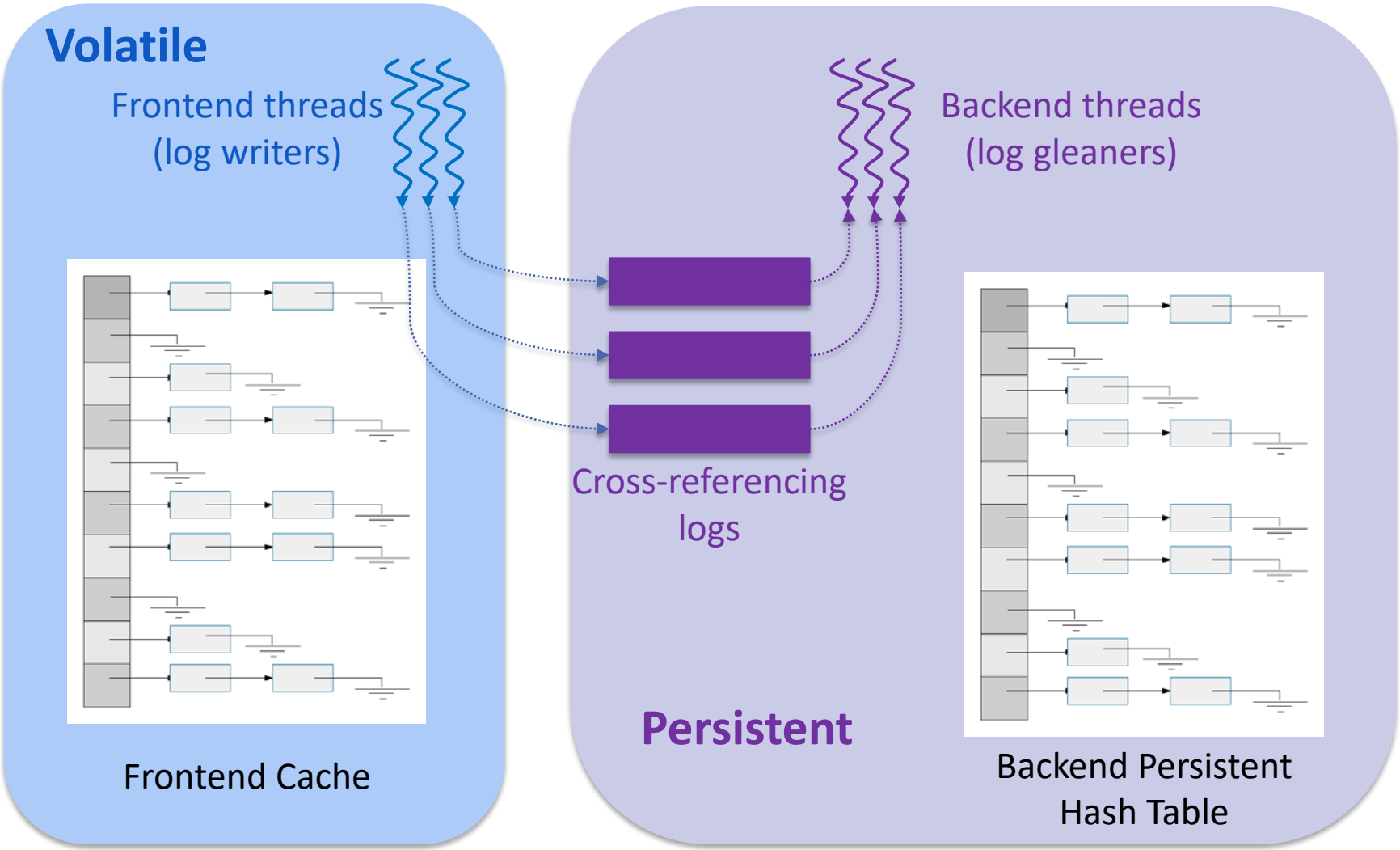
HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems
Fei Xia, *Institute of Computing Technology, Chinese Academy of Sciences; University of Chinese Academy of Sciences;* Dejun Jiang, Jin Xiong, and Ninghui Sun, *Institute of Computing Technology, Chinese Academy of Sciences*
Proceedings of the 2017 USENIX Annual Technical Conference, Santa Clara CA, July 2017.

Adaptation

Dynamic Worker Thread Switching



Closing the Performance Gap



Thank You!

I'm looking for Postdocs and students!

email me: mseltzer@cs.ubc.ca

Google

NSF National Science Foundation
WHERE DISCOVERIES BEGIN

NSERC
CRSNG