A Small Ride Towards Lock-Freedom

Miguel Areias
CRACS & INESC-TEC LA
Faculty of Sciences, University of Porto, Portugal

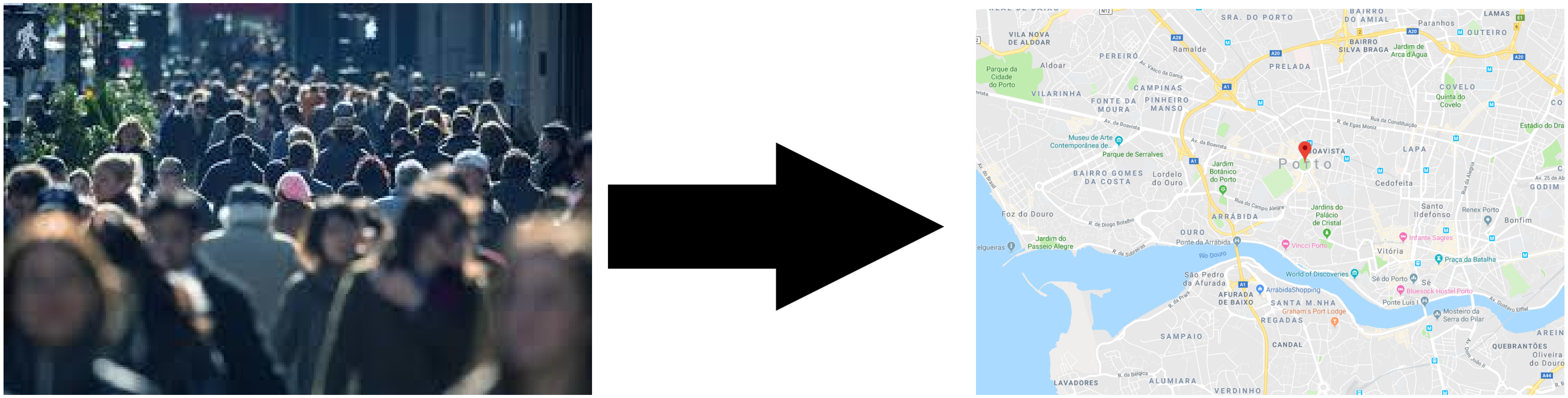
TALKS @ [dcc]

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Presentation Outline

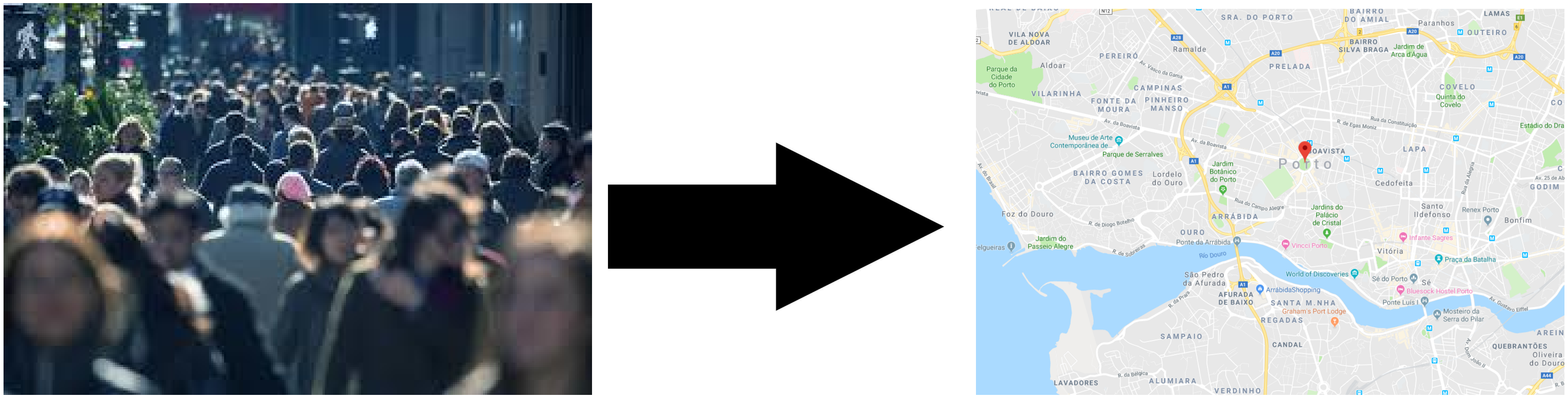
- Toy Example - Transportation Problem
- Concurrent Computing, Progress and Lock-Freedom
- Toy Example - Lock-Free Hash Map
- Questions & (Possible) Answers

Transportation Problem - Specifications



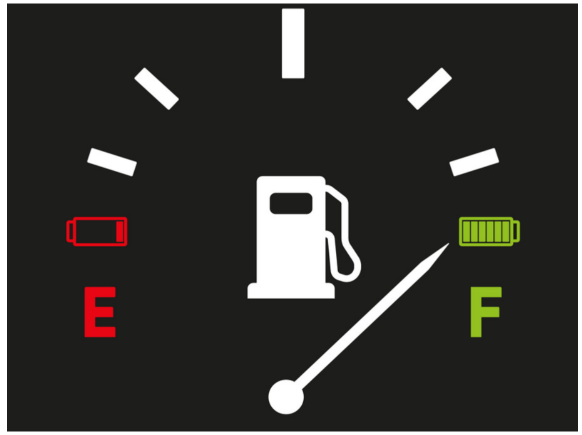
- Consider the **problem of transporting** the citizens of a city from multiples origins to multiple destinations. Specifications:
  - **One task** is **one transportation** of **one citizen** from a place A to a place B.
  - **One flow** is the **execution of one or more tasks**. It can be in one of two **states**: stopped or running.

Transportation Problem - Specifications



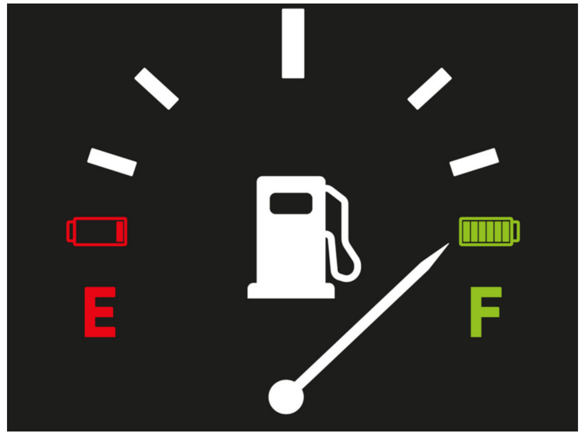
- Consider the **problem of transporting** the citizens of a city from multiples origins to multiple destinations. Specifications:
  - **One task** is **one transportation** of **one citizen** from a place A to a place B.
  - **One flow** is the **execution of one or more tasks**. It can be in one of two **states**: stopped or running.
  - A cold and severe entity called **environment**, **controls almost everything** about the city. For the **flows**, it can **control** their **state**, but it **cannot control** how they are **implemented**.

Transportation Problem - Environment

- What do we know about the **environment**.
  - It tries to be fair with **flows**. Tries to give them all of the necessary **resources** (good roads, gas, ...).
  - But, if the **flows** start demanding for more **resources** than the ones available, then huge traffic jams can occur.

Transportation Problem - Environment



- What do we know about the **environment**.
  - It tries to be fair with **flows**. Tries to give them all of the necessary **resources** (good roads, gas, ...).
  - But, if the **flows** start demanding for more **resources** than the ones available, then huge traffic jams can occur.
  - Thus, to avoid problems it uses the police to **control** the **flows**. The police can **interrupt** flows with almost arbitrary **"traffic stops"**. Once a **"traffic stop"** is complete, **flows** can resume their execution.
- How can we **implement** the **flows**?

Transportation Problem - Flows

- **Coarse granularity**.
- Join all tasks in a single **heavy flow F**.

Transportation Problem - Flows

- **Coarse granularity**.
- Join all tasks in a single **heavy flow F**.
- **Flow** management is simple:
  - **F** executes all tasks. **Traverses all origins/destinations** in the tasks until they are all completed.
  - No need to define extra policies.

Transportation Problem - Flows

- **Coarse granularity**.
- Join all tasks in a single **heavy flow F**.
- **Flow** management is simple:
  - **F** executes all tasks. **Traverses all origins/destinations** in the tasks until they are all completed.
  - No need to define extra policies.

- **One stoppage** can potentially affect all **tasks** (e.g. one traffic stop).

Transportation Problem - Flows

- **Middle granularity**.
- Join some tasks in **light flows**.

Transportation Problem - Flows

- **Middle granularity**.
- Join some tasks in **light flows**.
- **Flow** management is more complex:
  - **Different flows** can have different **origins/destinations**.
  - Must **define extra policies**:
    - join **tasks** by some criteria;
    - **manage** several **flows**;
    - ...

Transportation Problem - Flows

- **Middle granularity**.
- Join some tasks in **light flows**.
- **Flow** management is more complex:
  - **Different flows** can have different **origins/destinations**.
  - Must **define extra policies**:
    - join **tasks** by some criteria;
    - **manage** several **flows**;
    - ...
- **One stoppage** affects only the **tasks** within a **flow**.

Transportation Problem - Flows

- **Fine granularity**.
- Join some tasks in **even lighter flows**.

Transportation Problem - Flows

- **Fine granularity**.
- Join some tasks in **even lighter flows**.
- **Flow** management is more complex:
  - **Different flows** can have different **origins/destinations**.
  - ...
- **One stoppage** affects only the **tasks** within a **flow**.

Transportation Problem - Flows

- **Fine granularity**.
- Join some tasks in **even lighter flows**.
- **Flow** management is more complex:
  - **Different flows** can have different **origins/destinations**.
  - ...
- **One stoppage** affects only the **tasks** within a **flow**.
- So...the key idea is that, once you start **using multiple flows**, regardless of the **granularity**, you enter in a new world that has its own **particularities** (advantages and disadvantages).

Some Questions

- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)

Some Questions

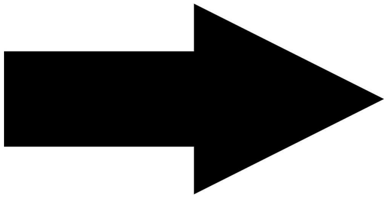
- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
- In this process, what can we **control**? Can we **control** the **environment**? (e.g. can we control the police?)

Some Questions

- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
- In this process, what can we **control**? Can we **control** the **environment**? (e.g. can we control the police?)
- How should we implement the **flows**? Having **multiple flows** should be better, right?

Some Questions

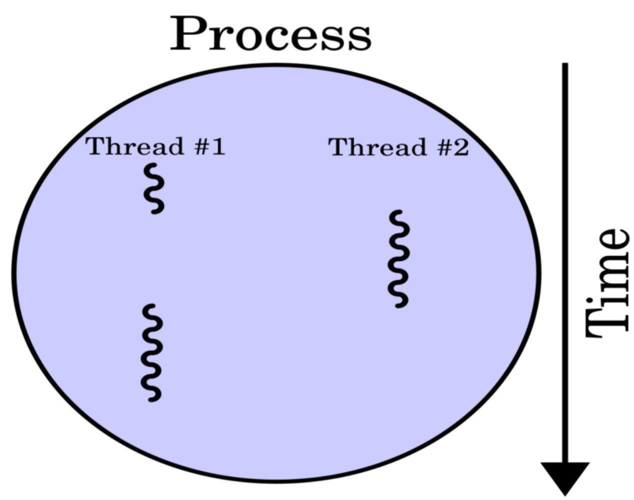
- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
- In this process, what can we **control**? Can we **control** the **environment**? (e.g. can we control the police?)
- How should we implement the **flows**? Having **multiple flows** should be better, right?
- Well...It depends. **Critical regions** are always a **problem** (e.g. crossroads).



U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

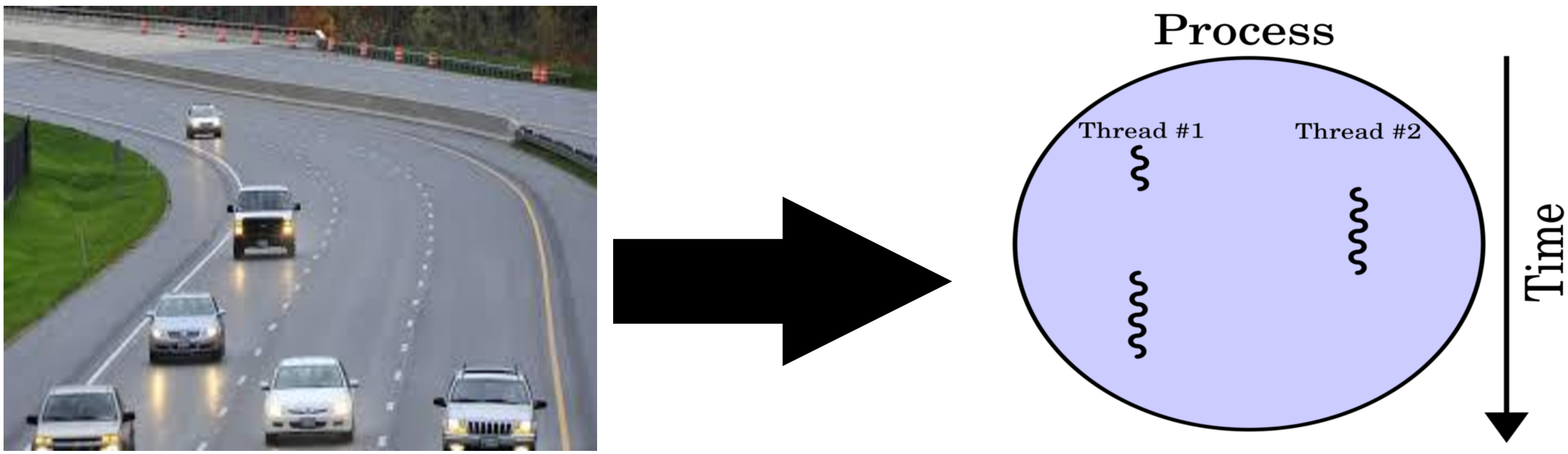
Transportation and Computing

- The word **thread** can be translated as **flow of control**.
- If we swap in the **toy example**:
  - the transportation **process** by an Operating System (OS) **process** and
  - consider that the **OS is the environment** (e.g., the OS scheduler could be the Police)



Transportation and Computing

- The word **thread** can be translated as **flow of control**.
- If we swap in the **toy example**:
  - the transportation **process** by an Operating System (OS) **process** and
  - consider that the **OS is the environment** (e.g., the OS scheduler could be the Police)



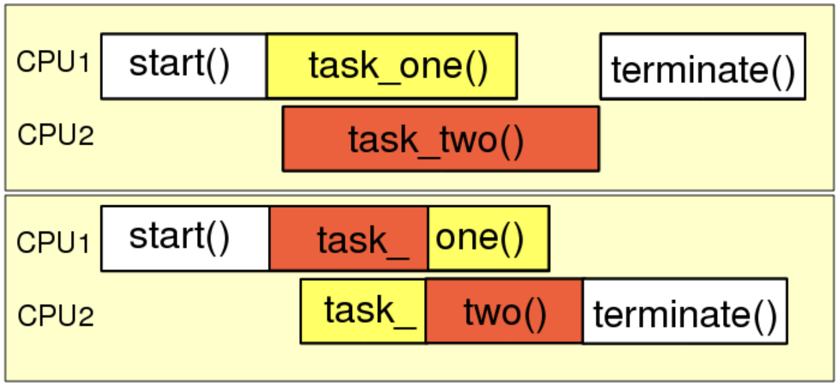
- then, we could be speaking about **parallel/concurrent** computing.

Parallel and Concurrent Computing

- **Parallel computing** occurs when a problem is decomposed in multiple parts that can be solved **concurrently**.

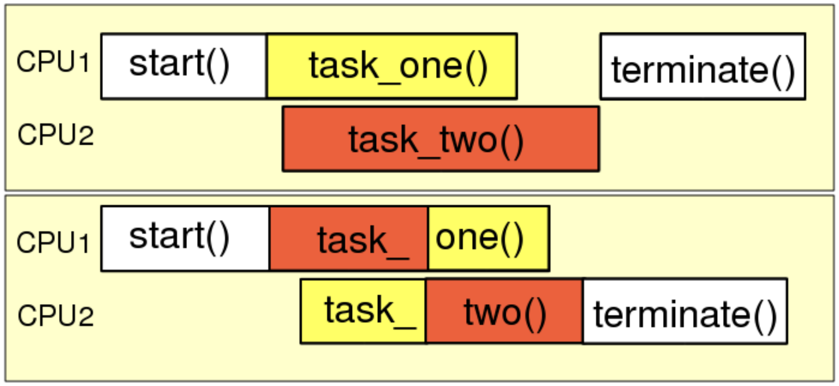
Parallel and Concurrent Computing

- **Parallel computing** occurs when a problem is decomposed in multiple parts that can be solved **concurrently**.
- A process exhibits **concurrency (or potential parallelism)** when it includes **tasks (contiguous parts of the process) that can be executed in any order** without **changing the expected result**.



Parallel and Concurrent Computing

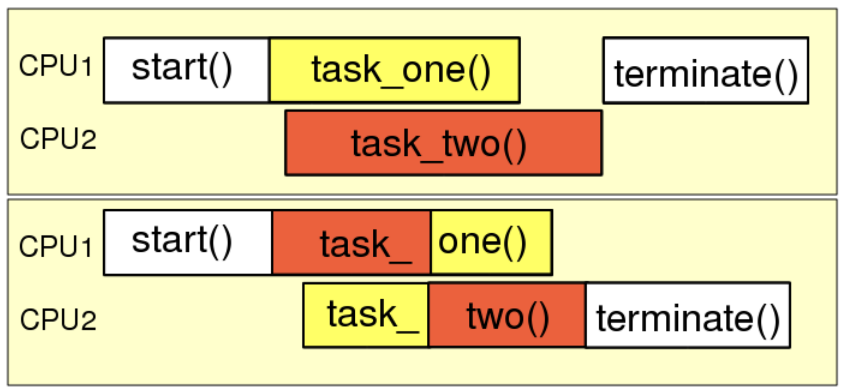
- **Parallel computing** occurs when a problem is decomposed in multiple parts that can be solved **concurrently**.
- A process exhibits **concurrency (or potential parallelism)** when it includes **tasks (contiguous parts of the process) that can be executed in any order** without **changing the expected result**.



- How do we know that a **concurrent computation** will converge towards the **termination**?

Parallel and Concurrent Computing

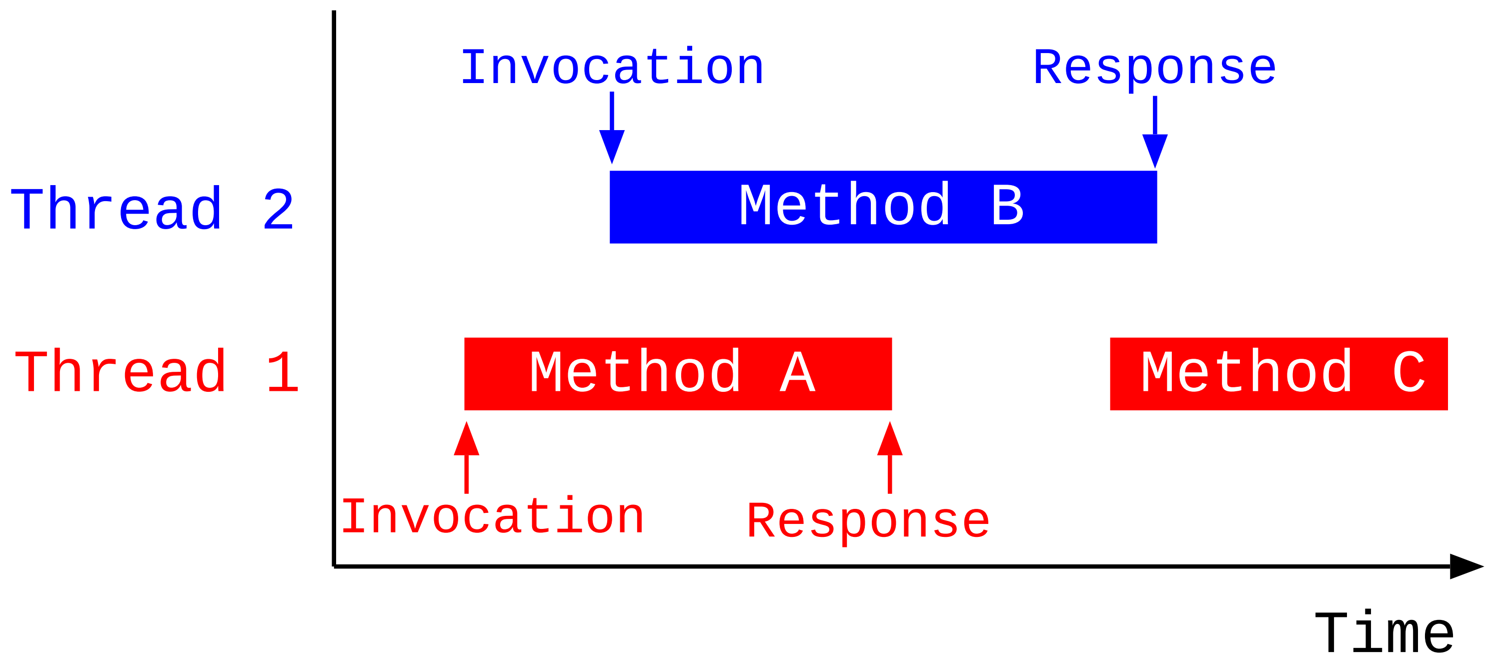
- **Parallel computing** occurs when a problem is decomposed in multiple parts that can be solved **concurrently**.
- A process exhibits **concurrency (or potential parallelism)** when it includes **tasks (contiguous parts of the process) that can be executed in any order** without **changing the expected result**.



- How do we know that a **concurrent computation** will converge towards the **termination**?
  - We must check its **progress properties**.

Progress

- In 2011, Herlihy and Shavit presented a **grand unified explanation** for the **progress properties**. **Progress** is seen as the **number of steps** that threads take to **complete methods** within a concurrent object, i.e., the **number of steps** that threads take to **execute methods** between their **invocation** and their **response**.



Progress

- **Progress** models are placed in a two-dimensional **periodical table**, where the two axes define the:
  - **dependency** on the **operating system (OS) scheduler**.
  - **level of progress** provided by the methods.

Dependency on the operating system scheduler

Dependency vs Progress	Non-Blocking Independent	Non-Blocking Dependent	Blocking Dependent
Every thread makes progress	Wait-Free	Obstruction-Free	Starvation-Free
Some thread makes progress	Lock-Free	?	Deadlock-Free

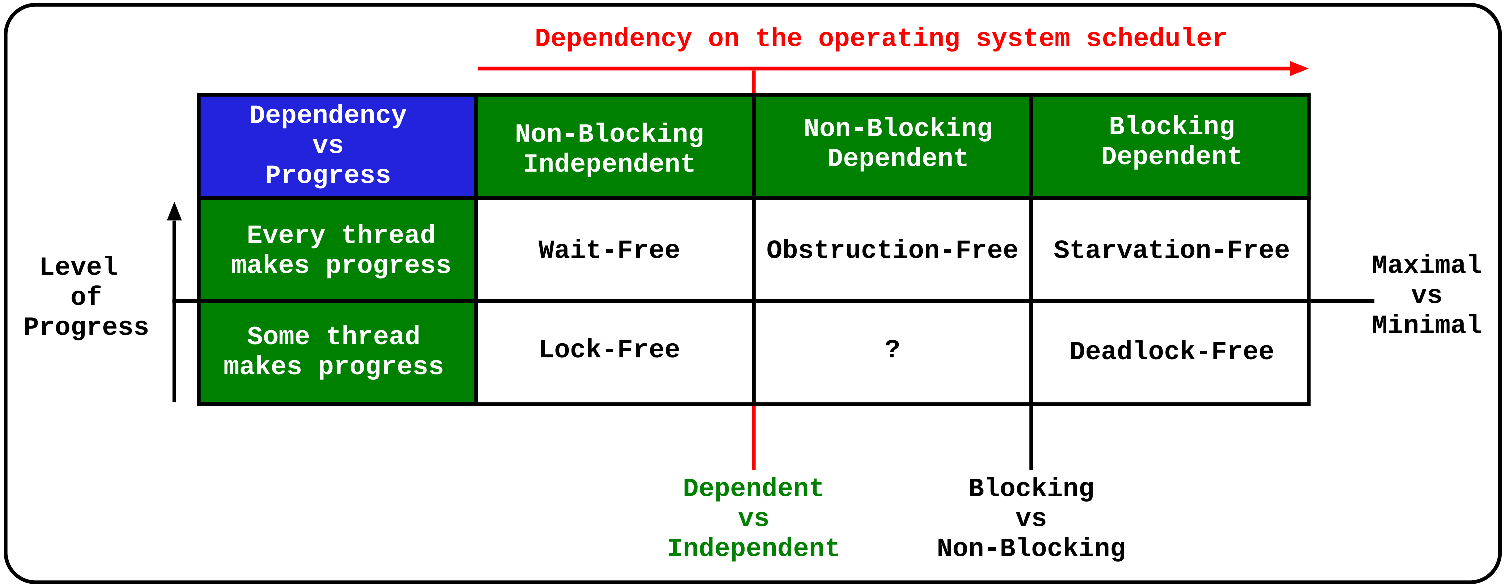
Level of Progress

Maximal vs Minimal

Dependent vs Independent

Blocking vs Non-Blocking

Progress



- For the **dependency**, a scheduler:
  - **dependent** model, means that the **progress** of threads relies on the **OS scheduler** to satisfy certain properties.
  - **independent** model, means that threads **progress** as long as they **are scheduled** (does not matter how).

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Progress

Dependency on the operating system scheduler

Dependency vs Progress	Non-Blocking Independent	Non-Blocking Dependent	Blocking Dependent
Every thread makes progress	Wait-Free	Obstruction-Free	Starvation-Free
Some thread makes progress	Lock-Free	?	Deadlock-Free

Level of Progress

Maximal vs Minimal

Dependent vs Independent

Blocking vs Non-Blocking

➤ For the **level of progress**, a method provides:

- ♦ **minimal progress**, if a thread calling that method can take an **infinite number of steps** without returning.
- ♦ **maximal progress**, if a thread calling that method takes a **finite number of steps** to return.

Progress

Dependency on the operating system scheduler

Dependency vs Progress	Non-Blocking Independent	Non-Blocking Dependent	Blocking Dependent
Every thread makes progress	Wait-Free	Obstruction-Free	Starvation-Free
Some thread makes progress	Lock-Free	?	Deadlock-Free

Level of Progress

Maximal vs Minimal

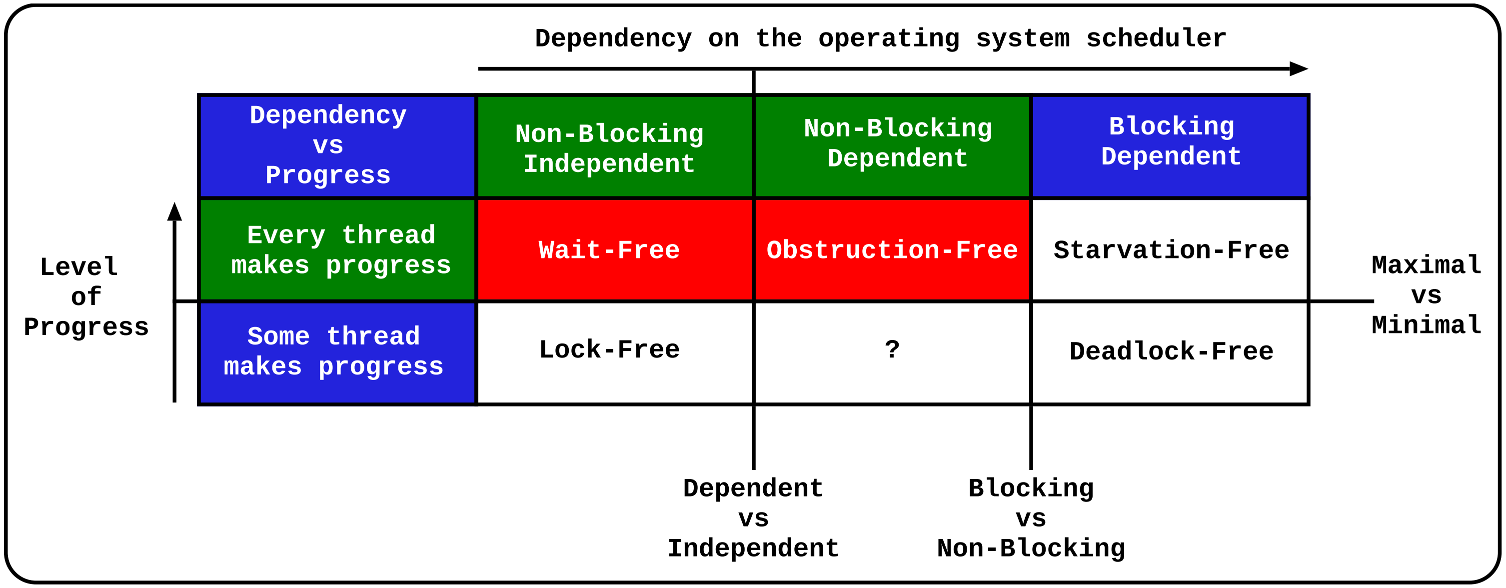
Dependent vs Independent

Blocking vs Non-Blocking

- **Starvation-Free**: a critical region cannot be denied to a thread perpetually.
- **Deadlock-Free**: a thread cannot delay other threads perpetually.
- Both, **rely** on the assumption that, the **OS scheduler allows** a thread within a **critical region**, to be **able to run** for **sufficient amount** of time, such that it can **leave** that **critical region** (**blocking dependent**).

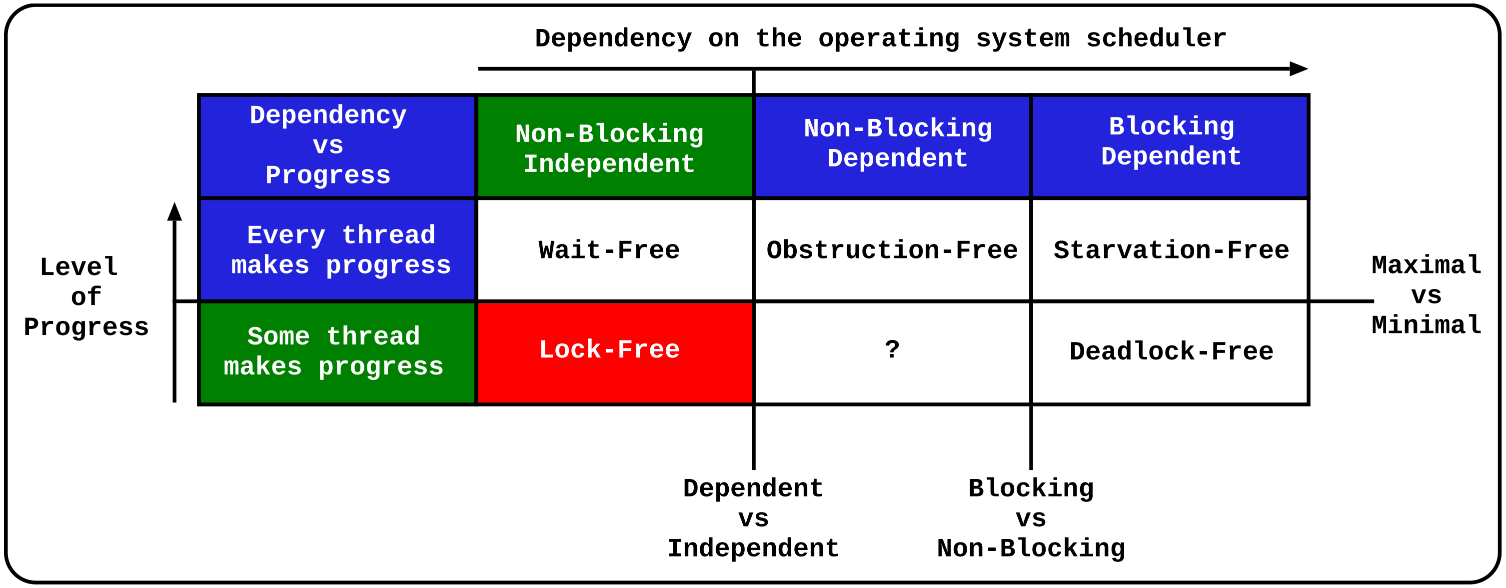
U.PORTO FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Progress



- **Obstruction-free** (a thread runs within a critical region in a bounded number of steps) **relies** on the assumption that the **OS scheduler** allows a thread to **run in isolation** for a **sufficient** amount of time (**non-blocking dependent**).
- **Wait-free** (a thread is able to make progress in a finite number of steps) provides **maximal progress** and has **no requirements** on the **OS scheduler** (**non-blocking independent**).

Progress



- **Lock-free** provides **minimal progress** and has **no requirements** on the **OS scheduler** (**non-blocking independent**).
- **Lock-free** guarantees then that, on **every instant** of the execution of methods (between their invocation and their response), **at least one thread** is doing **progress** on its work.

U.PORTO FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

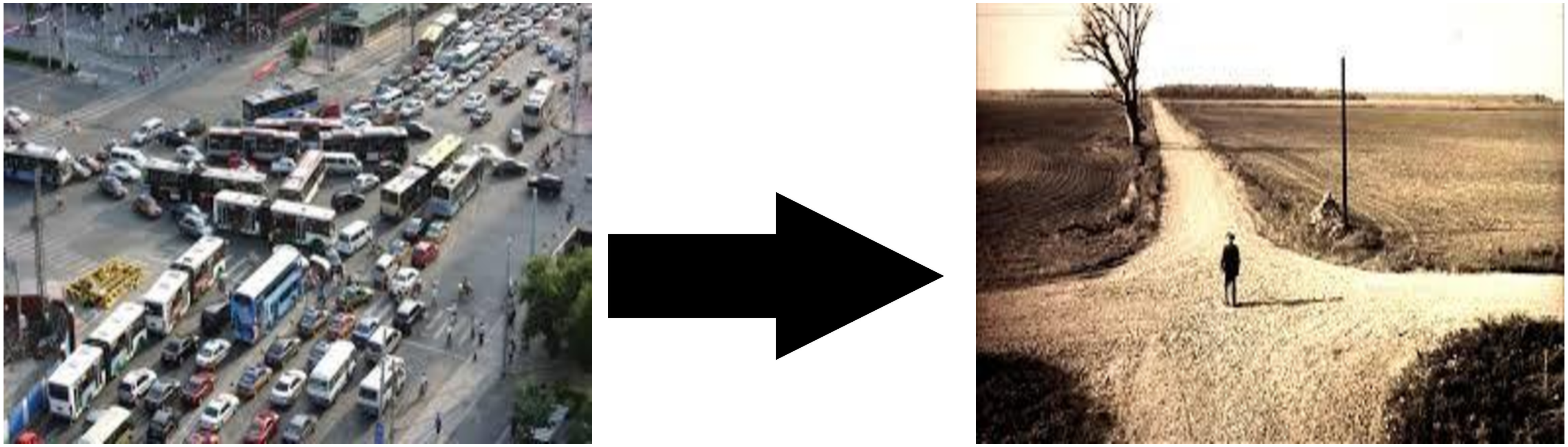
Lock-Freedom

- **Lock-Free objects** allow **greater concurrency** than **lock-based objects** since **semantically consistent** (non-interfering) methods **may execute in parallel**.
- **Lock-Free** techniques **do not use** traditional **locking** mechanisms.
  - **Avoid problems** such as:
    - **deadlocks** - threads **delaying** each other **perpetually**.
    - **convoying** - a thread holding a lock is **descheduled** by an **interrupt**.
    - **kill-intolerant** - a thread **is not immune** to the **dead** of other threads during the execution.

Lock-Freedom

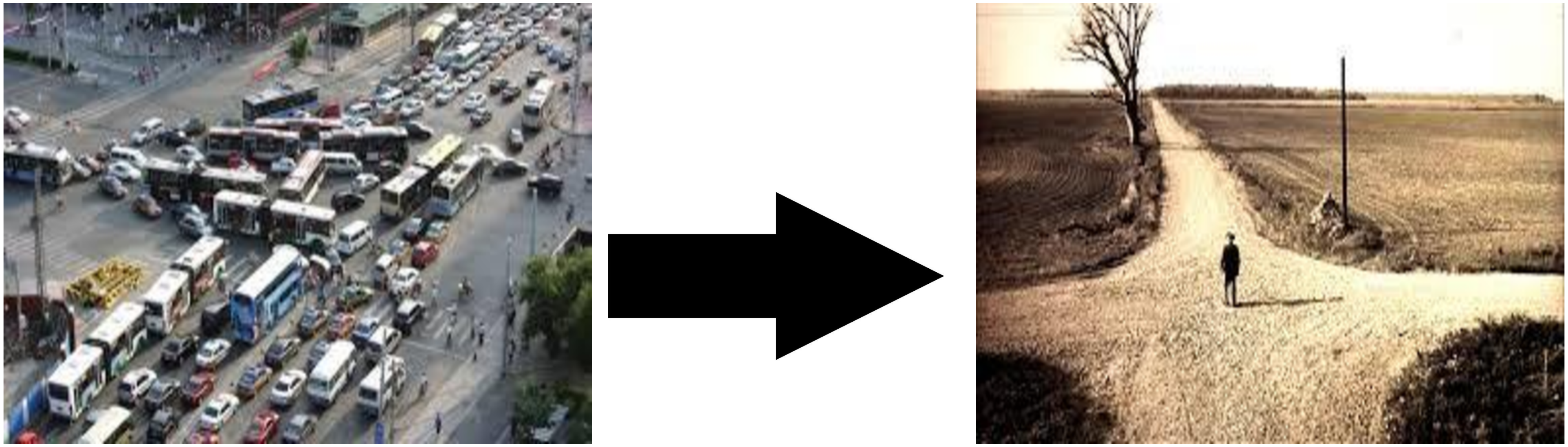
- **Lock-Free objects** allow **greater concurrency** than **lock-based objects** since **semantically consistent** (non-interfering) methods **may execute in parallel**.
- **Lock-Free** techniques **do not use** traditional **locking** mechanisms.
  - **Avoid problems** such as:
    - **deadlocks** - threads **delaying** each other **perpetually**.
    - **convoying** - a thread holding a lock is **descheduled** by an **interrupt**.
    - **kill-intolerant** - a thread **is not immune** to the **dead** of other threads during the execution.
    - **priority inversion** - a thread with **high priority** is **preempted** by a thread with **lower priority**.
    - **contention** - a thread **waiting** for a lock that is being **held by another** thread.

Lock-Freedom



- They are **based** in placing simple **atomic operations** in key **concurrency spots**, to **improve performance** and **ensure correctness** (formal proof of **linearization**).
  - **Atomic** operations **cannot** be interrupted (intrinsically **thread safe**).

Lock-Freedom

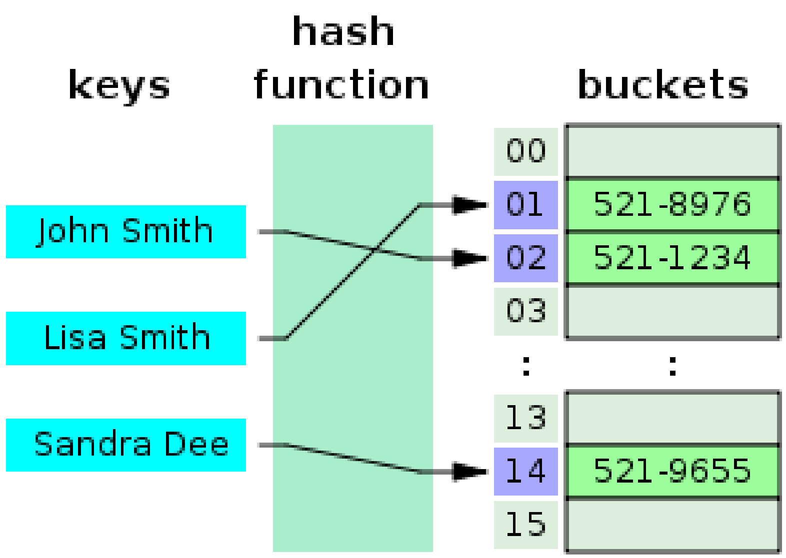


- At the **implementation level**, they take advantage of the **CAS (Compare-and-Swap) atomic operation**, that nowadays **can be found** in many common **hardware** architectures.
    - **CAS**(**Memory_Reference**, **Expected_Value**, **New_Value**).

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Toy Example - Hash Maps

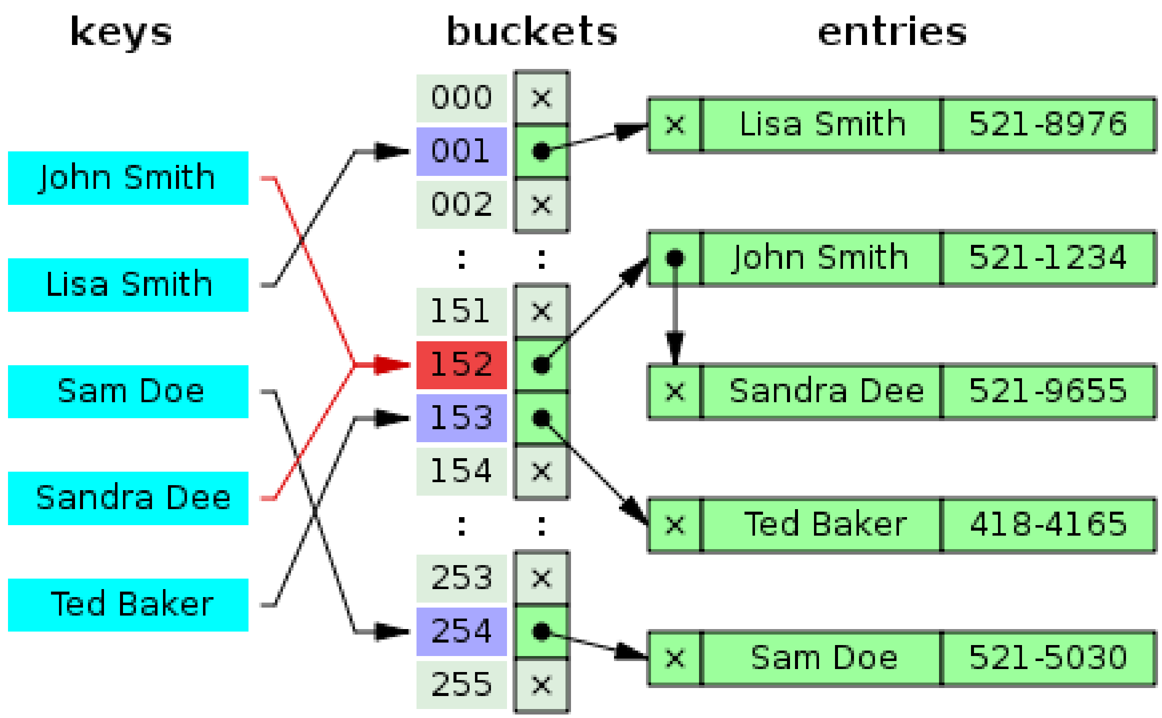
- **Hash maps** are **useful** to **store information** that can be organized as **pairs (K, C)**, where **K** is an identifier (or a **key**) and **C** is the **associated content**.



A small phone book as a hash map.

Hash Maps

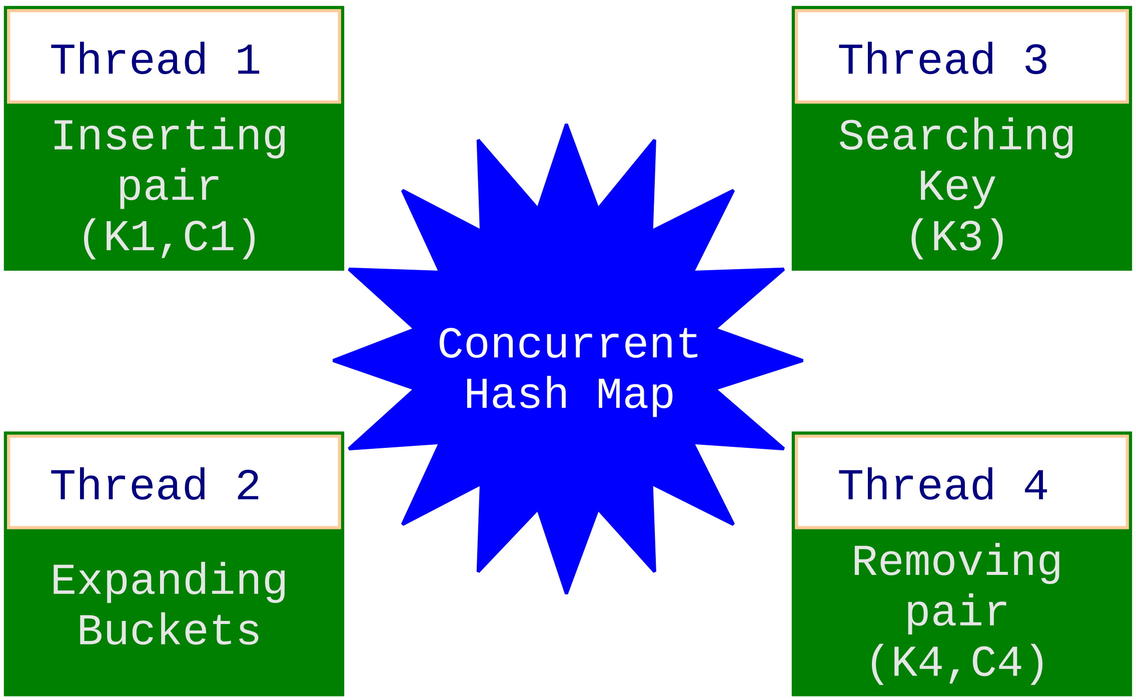
- Some of the most **usual methods** are:
  - **User-level** (externally activated by users) : **search**, **insert** and **remove**.
  - **Kernel-level** (internally activated by thresholds): **expansion** (key collision) (and **compression**, which will not be discussed in this talk).



Key collisions resolved using a separate chaining mechanism.

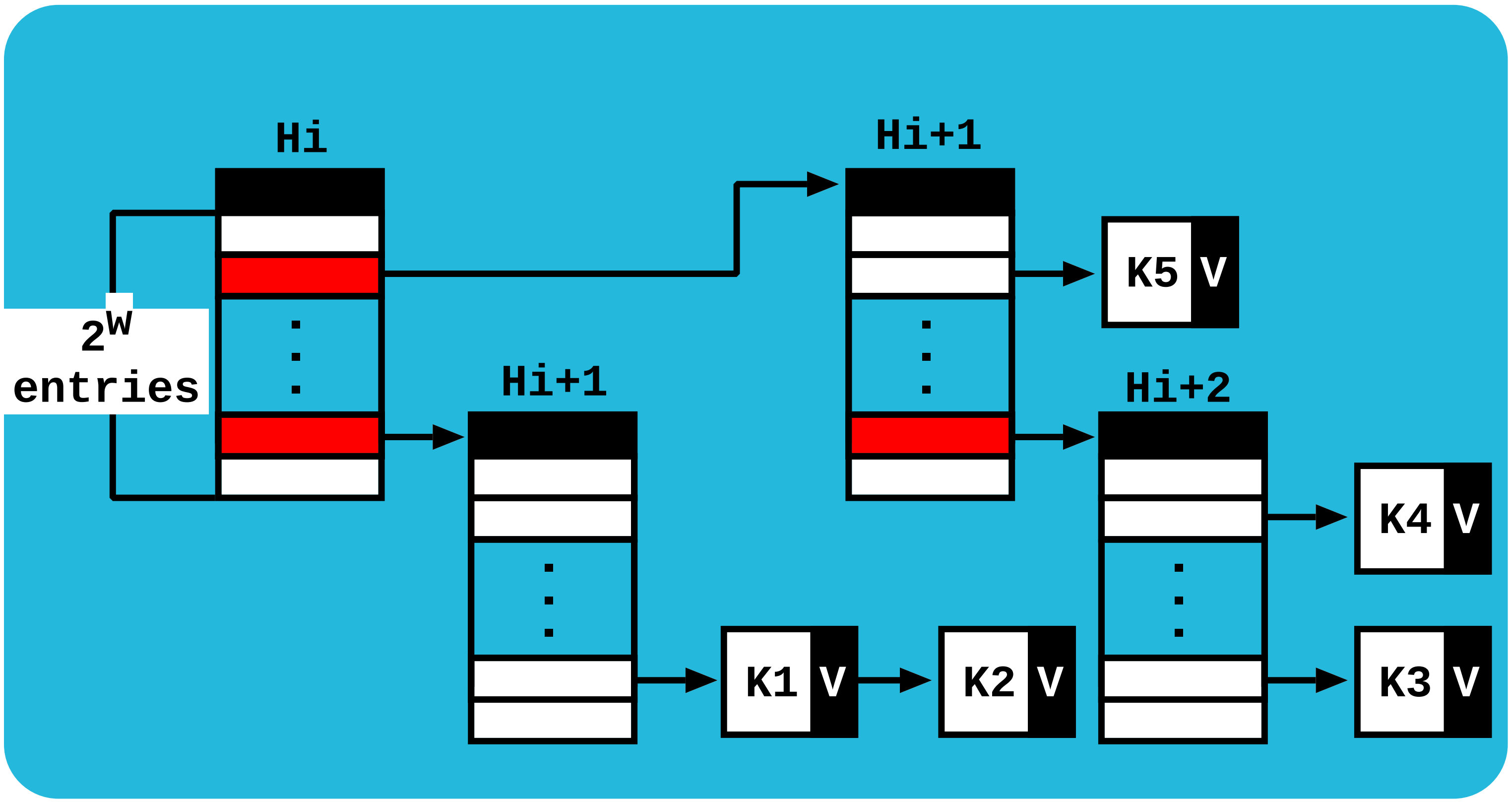
Concurrent Hash Maps

- Allow the **concurrent execution** of multiple **methods**.
  - **Each method** runs **independently**, but at the engine level, **all methods share** the underlying **data structures** that support the **hash map**.

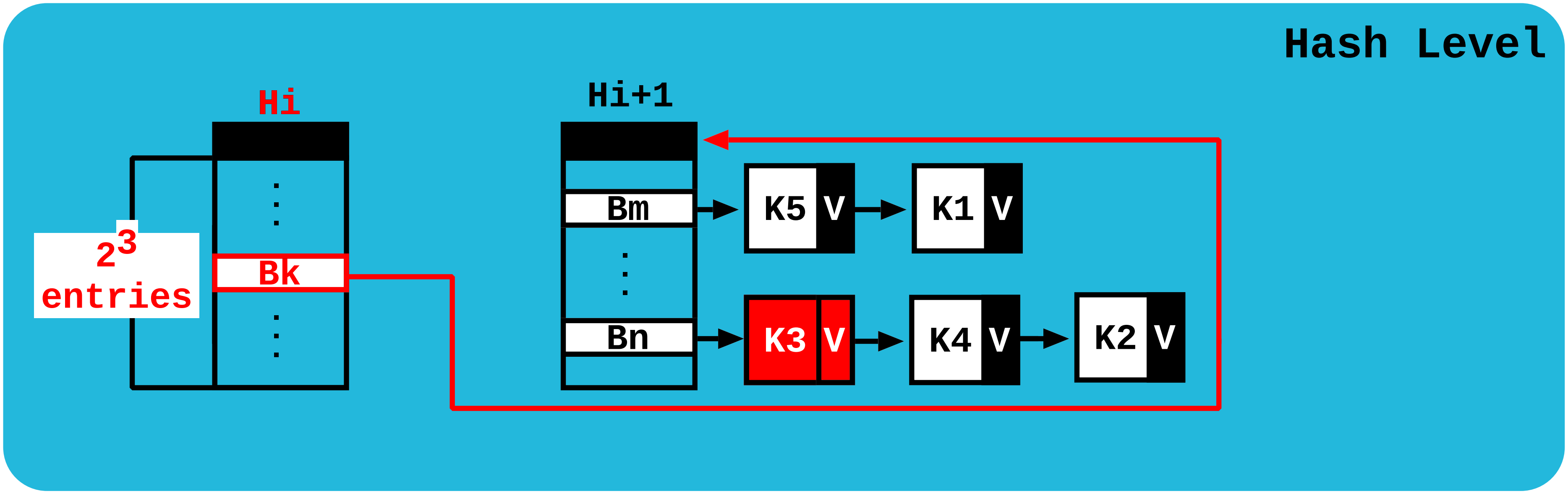


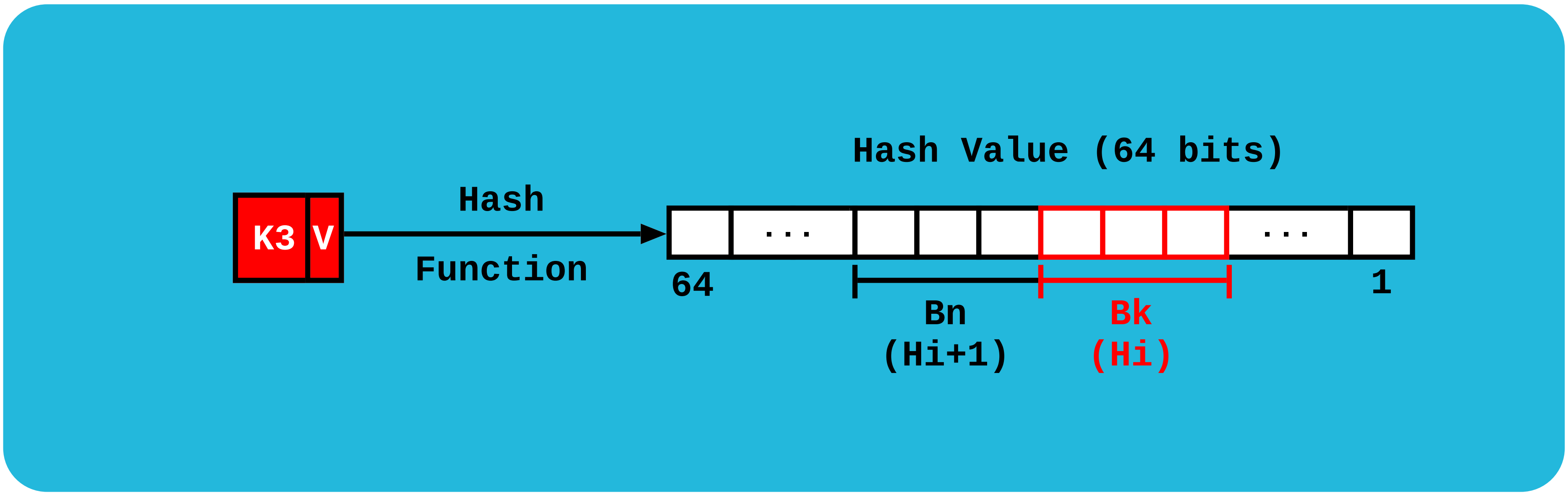
The LF Design - Hash Trie Structure

- **Hash buckets** refer to a **chaining mechanism** that supports **key collisions**.
- **Chain nodes** store **pairs (Key, Content, (Next_On_Chain, State))**. For the **sake of simplicity** we will present only **(Key, (Next_On_Chain, State))**. **State** can be **valid (V)** or **invalid (I)**.

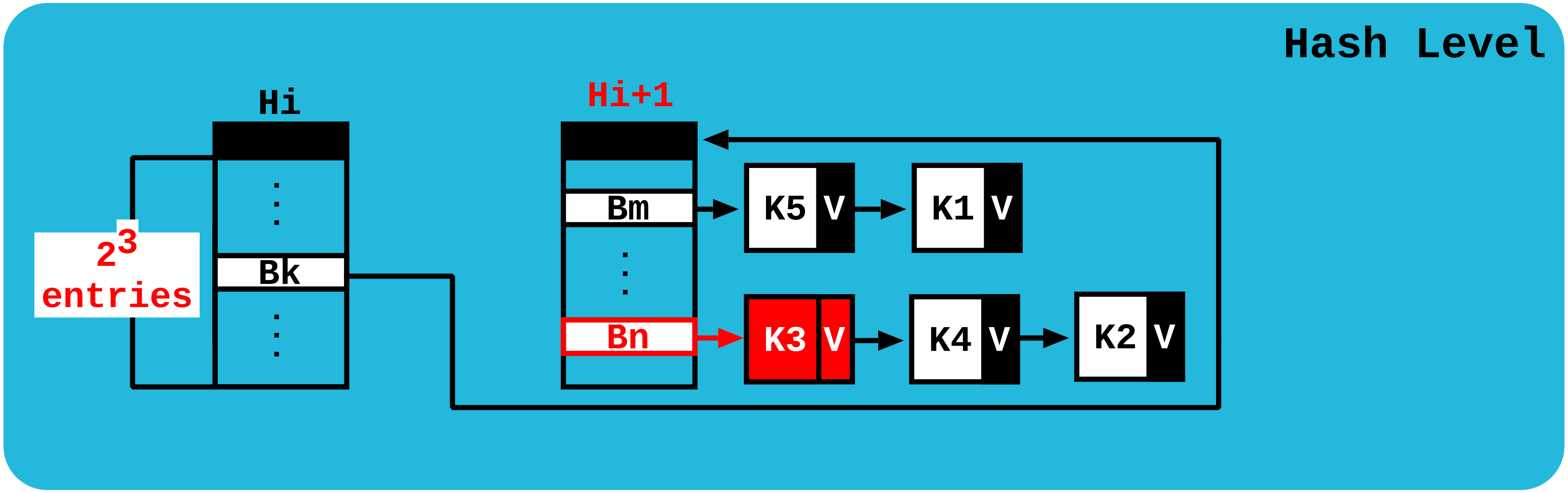


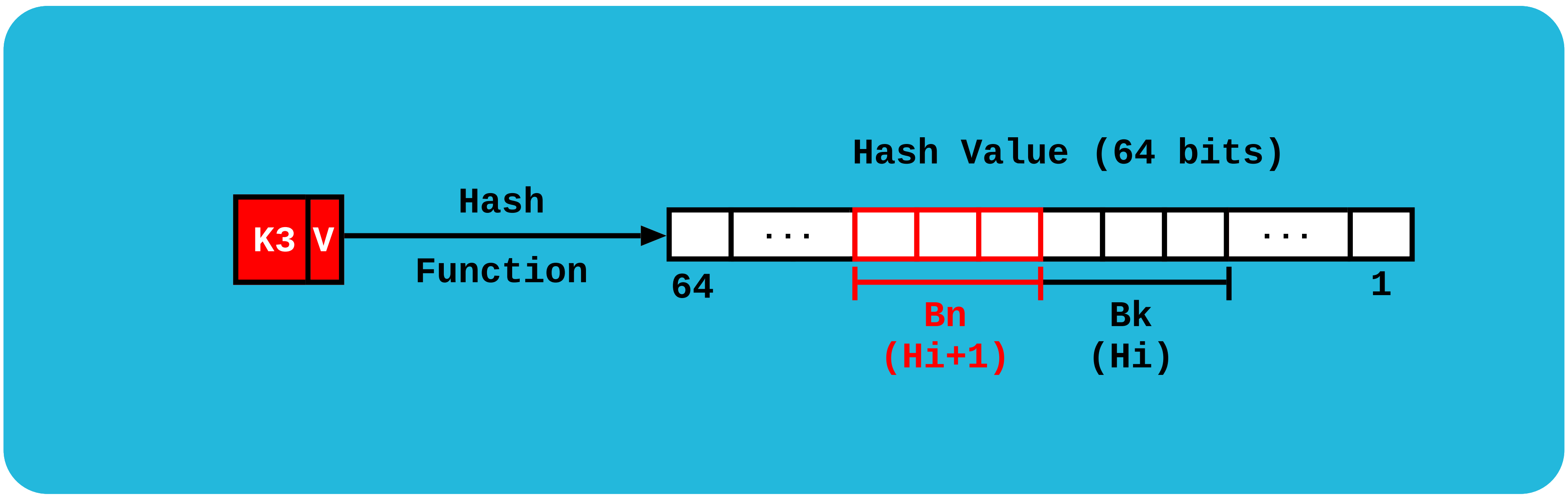
The LF Design - Searching for K3





The LF Design - Searching for K3



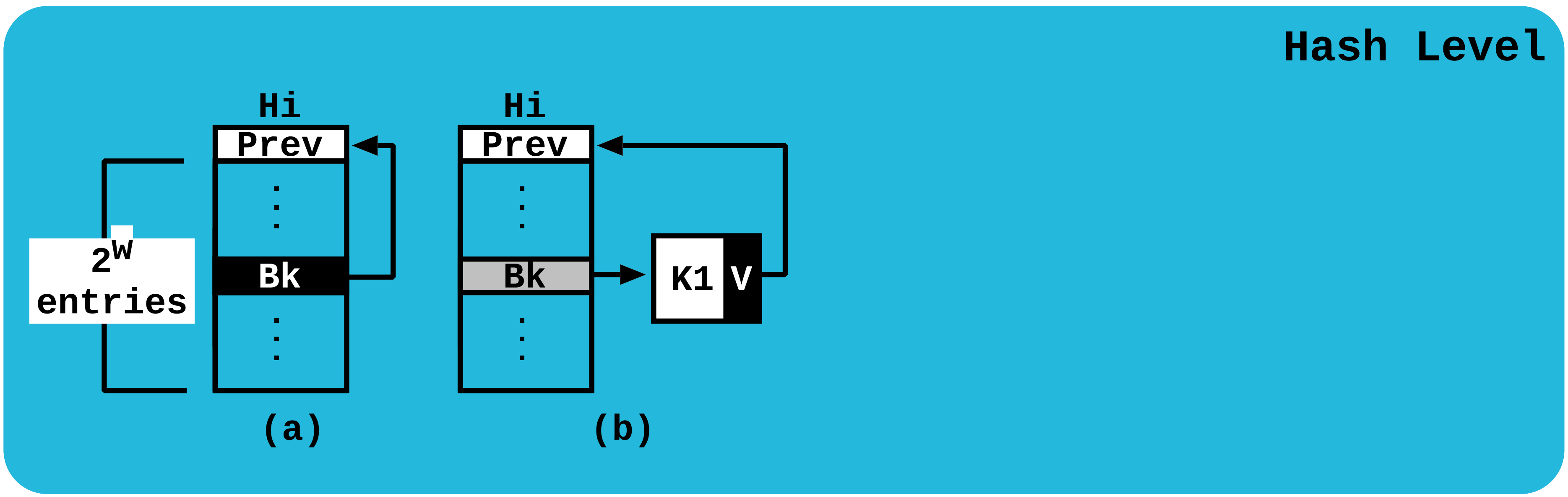


The LF Design - Internals

- To support **concurrency**, our design allows **threads** to:
  - **Recover from preemption**, by using a **Prev** field to traverse the **hash buckets** backwards.
  - **Identify chains**, by using a **back-reference** on the end of each chain.
  - **Maintain consistency**, by using **CAS** on write operations.

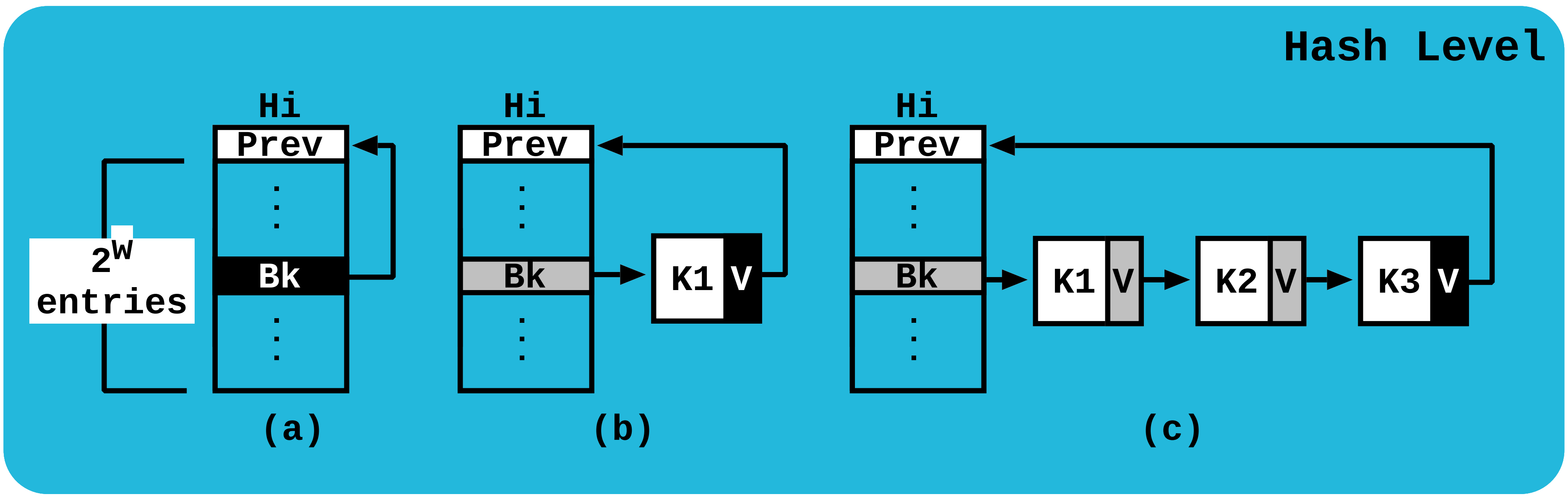
The LF Design - Internals

- To support **concurrency**, our design allows **threads** to:
  - **Recover from preemption**, by using a **Prev** field to traverse the **hash buckets** backwards.
  - **Identify chains**, by using a **back-reference** on the end of each chain.
  - **Maintain consistency**, by using **CAS** on write operations.

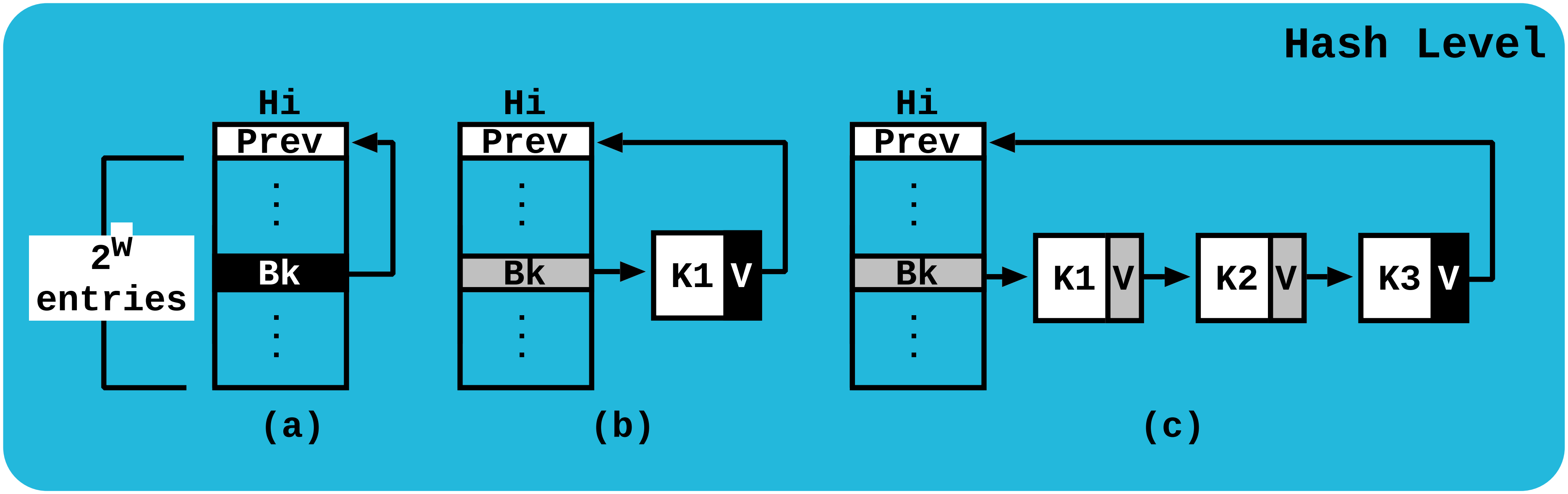


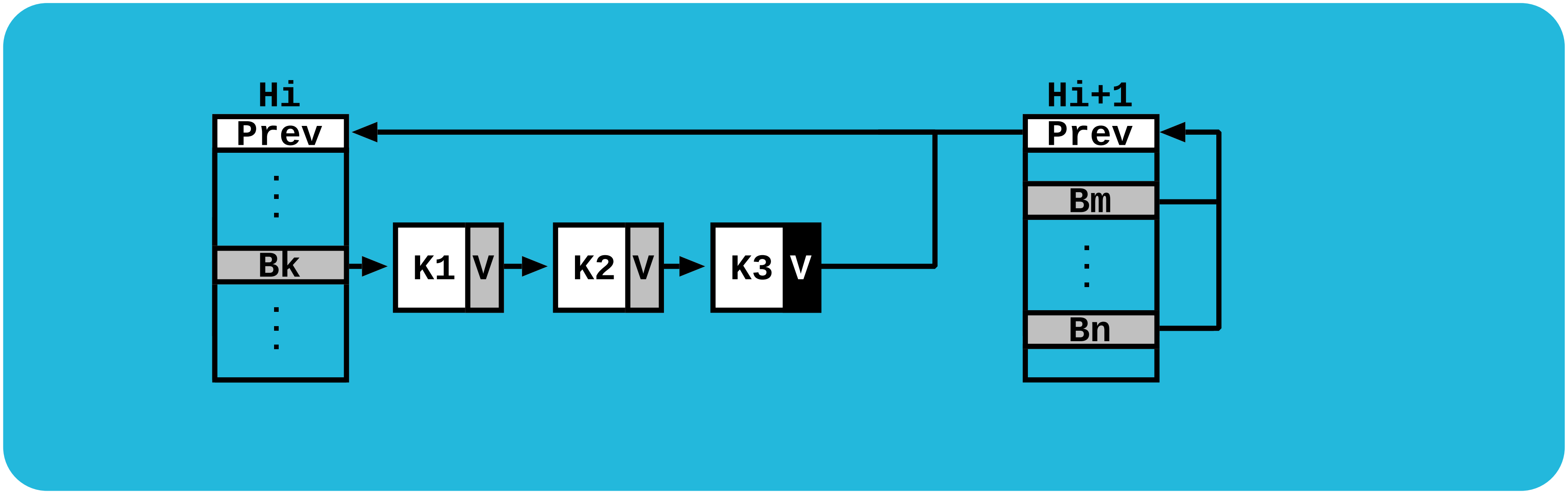
The LF Design - Internals

- To support **concurrency**, our design allows **threads** to:
  - **Recover from preemption**, by using a **Prev** field to traverse the **hash buckets** backwards.
  - **Identify chains**, by using a **back-reference** on the end of each chain.
  - **Maintain consistency**, by using **CAS** on write operations.

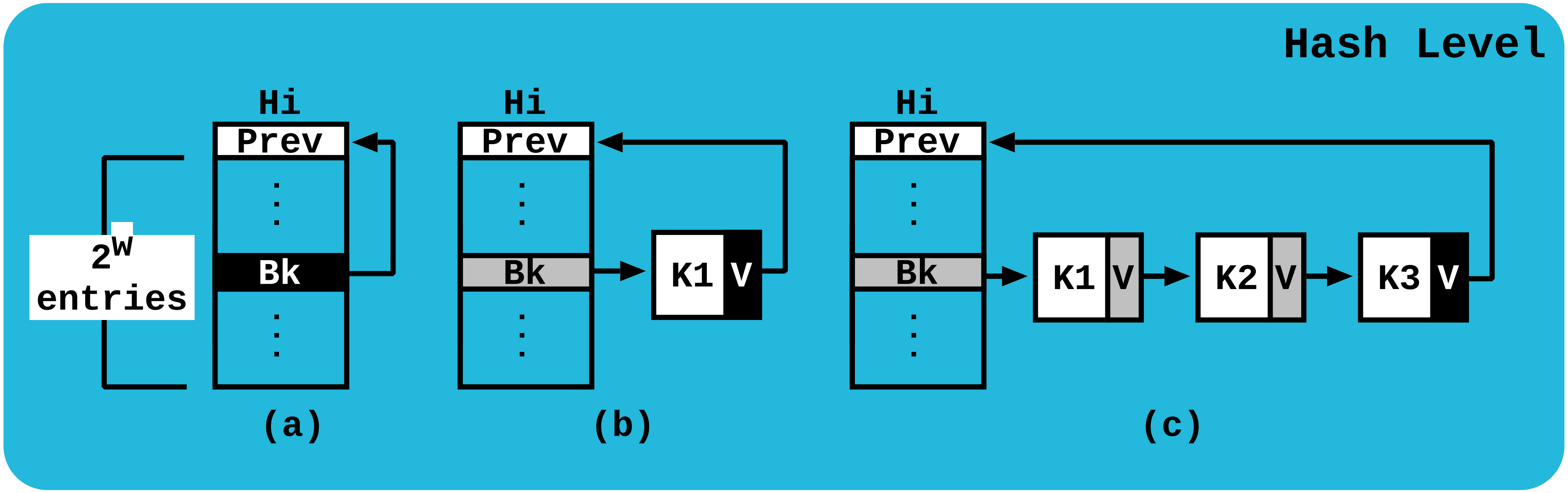


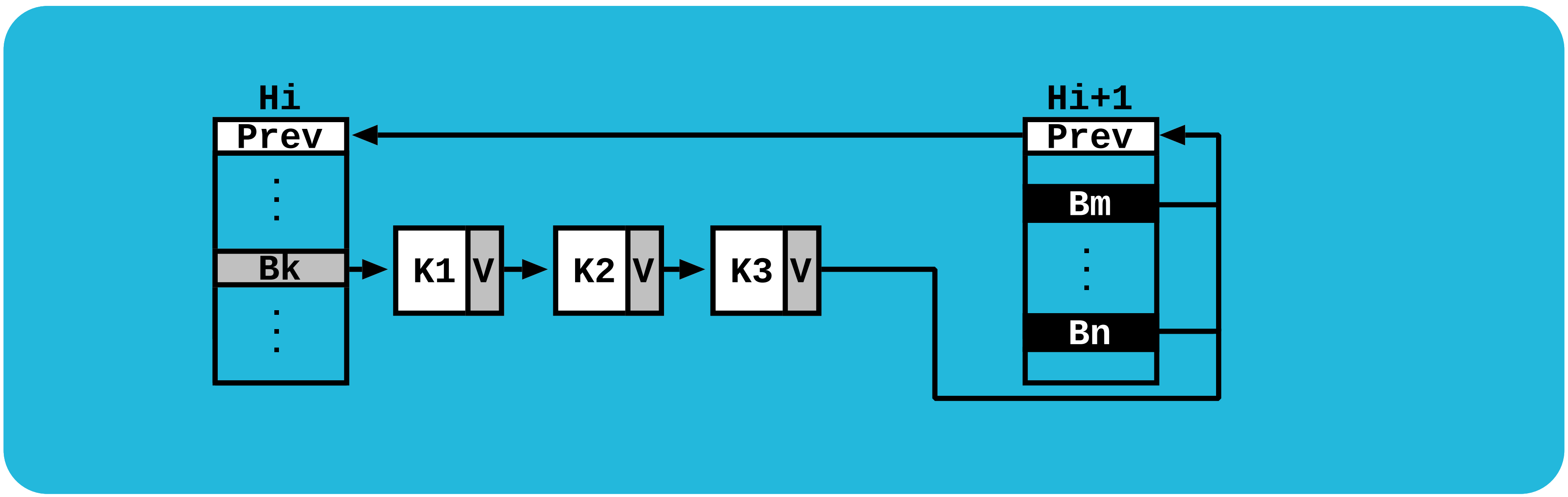
The LF Design



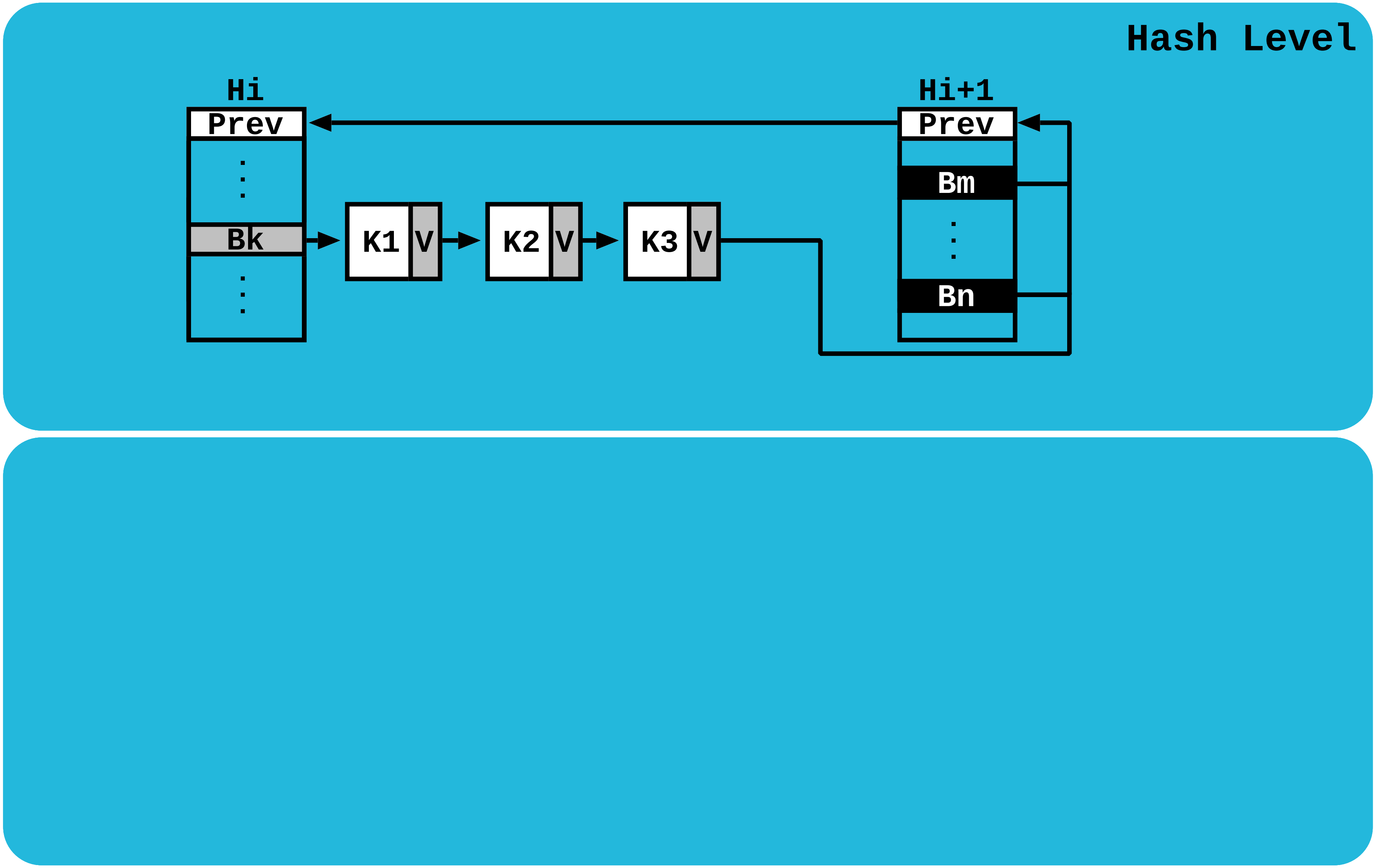


The LF Design

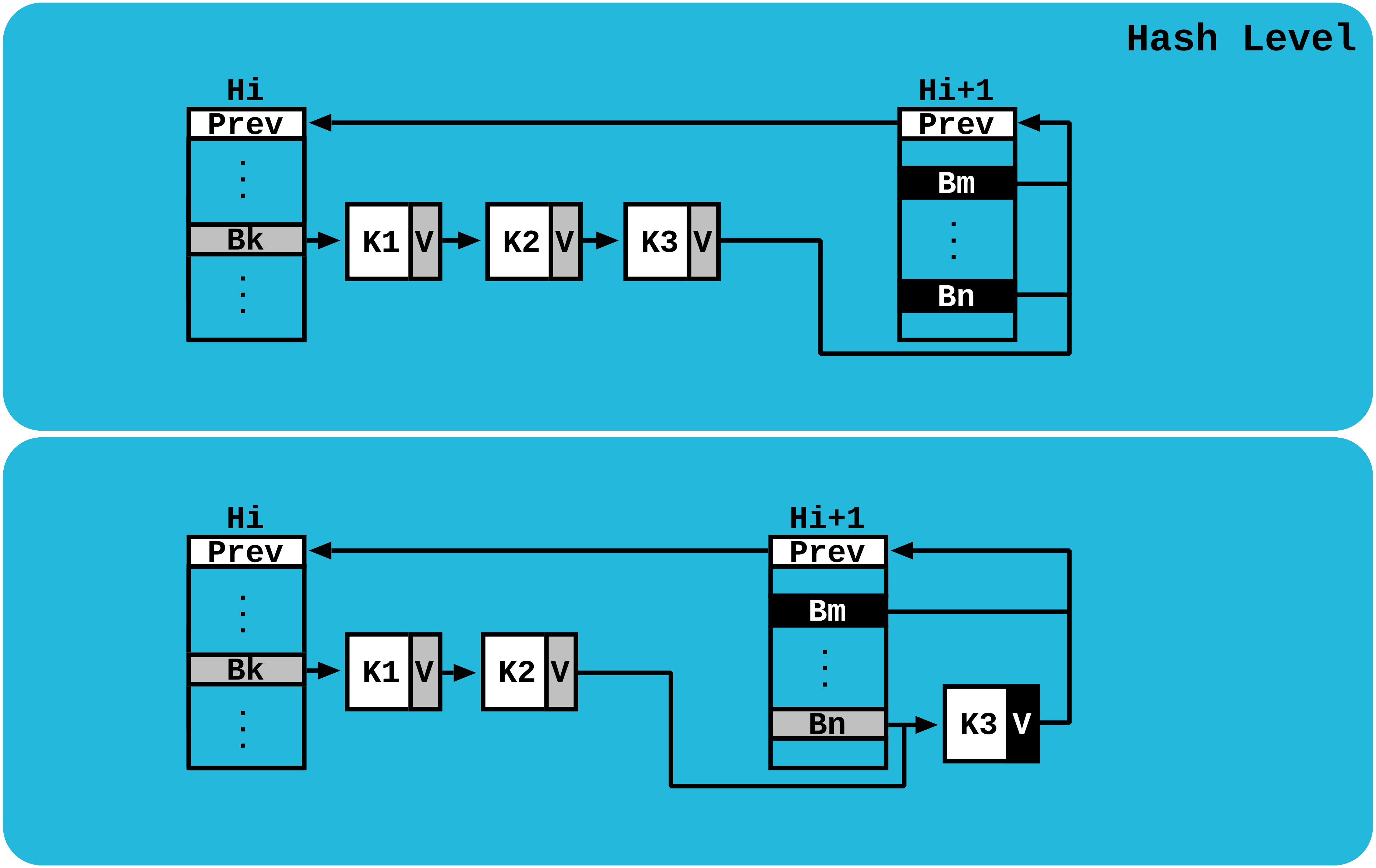




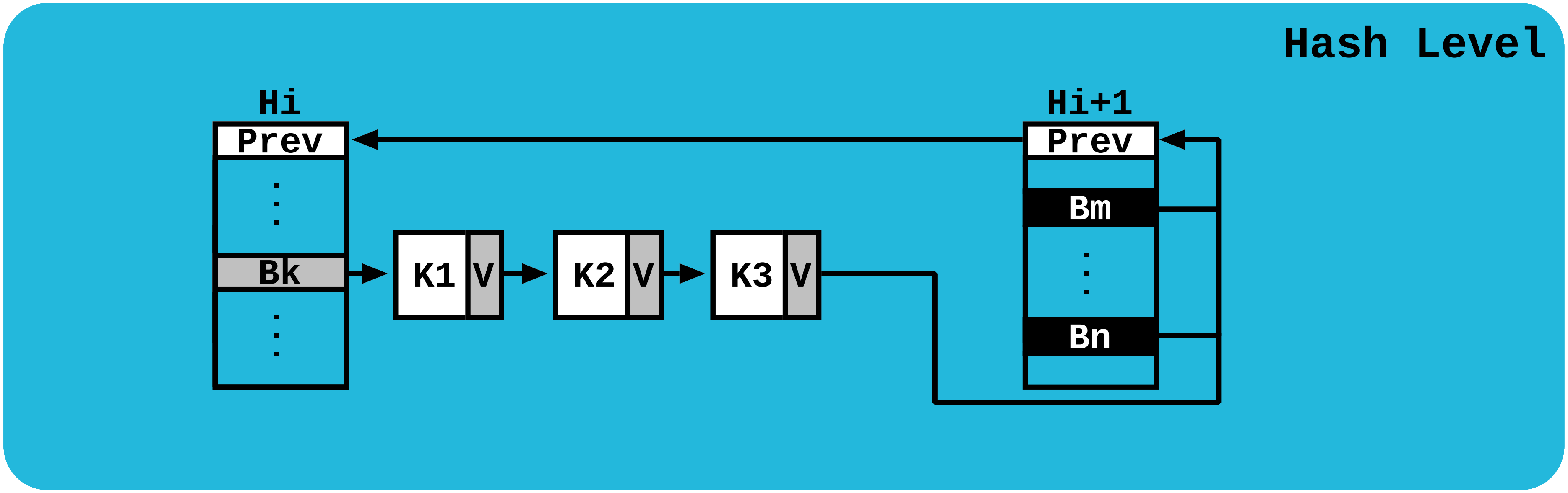
The LF Design

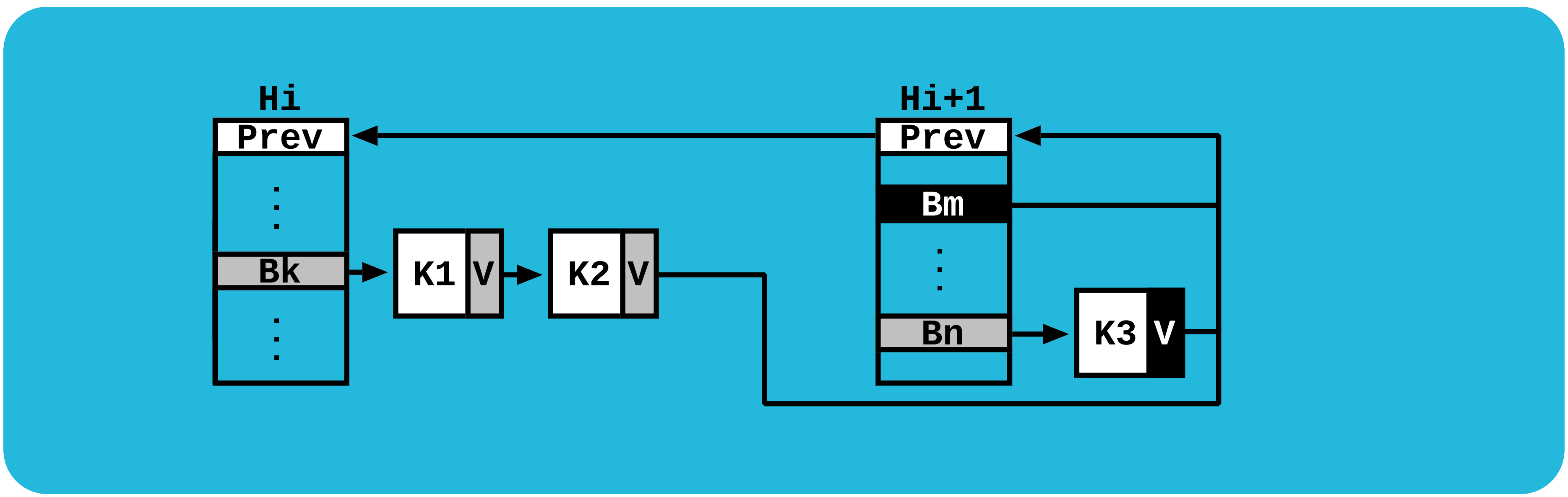


The LF Design

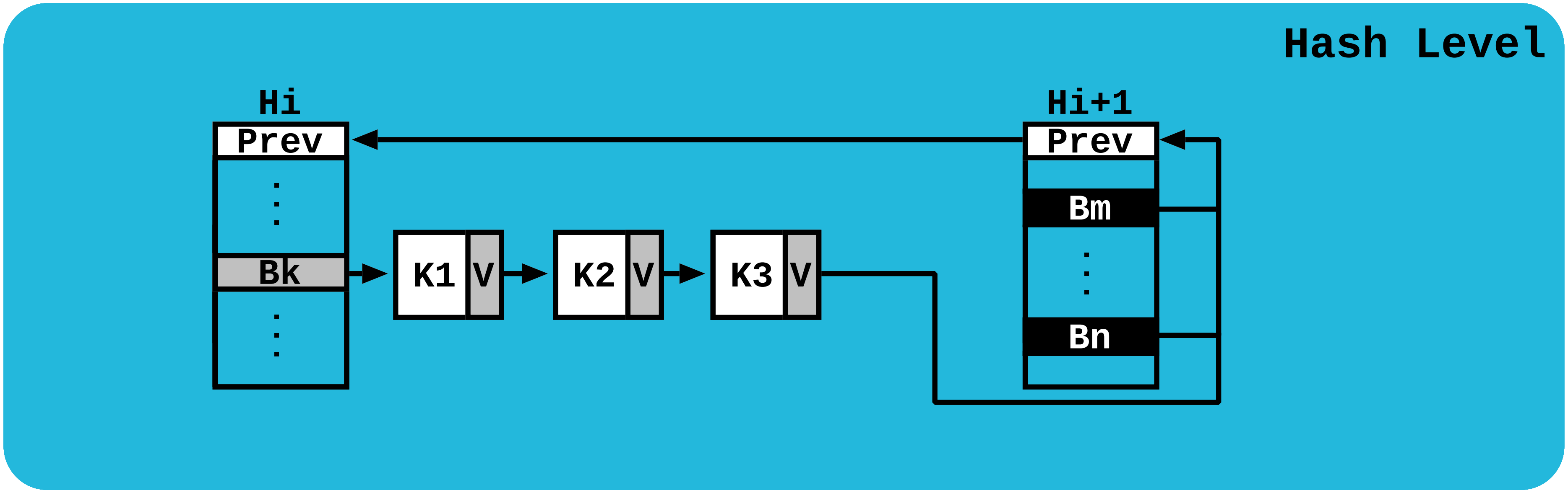


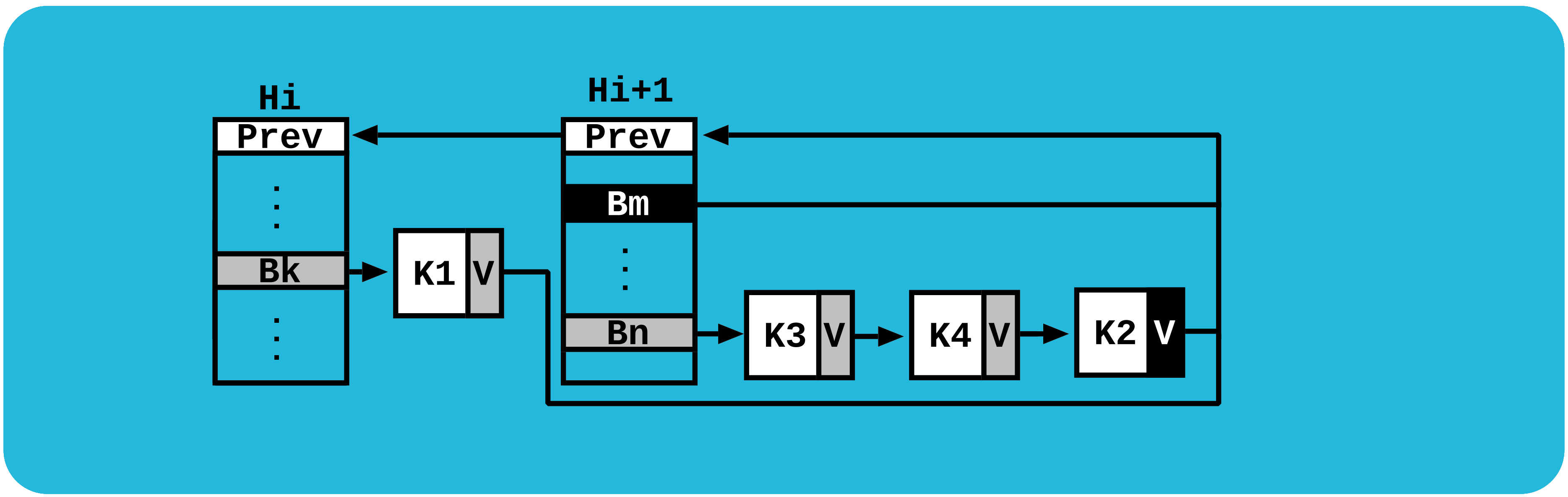
The LF Design



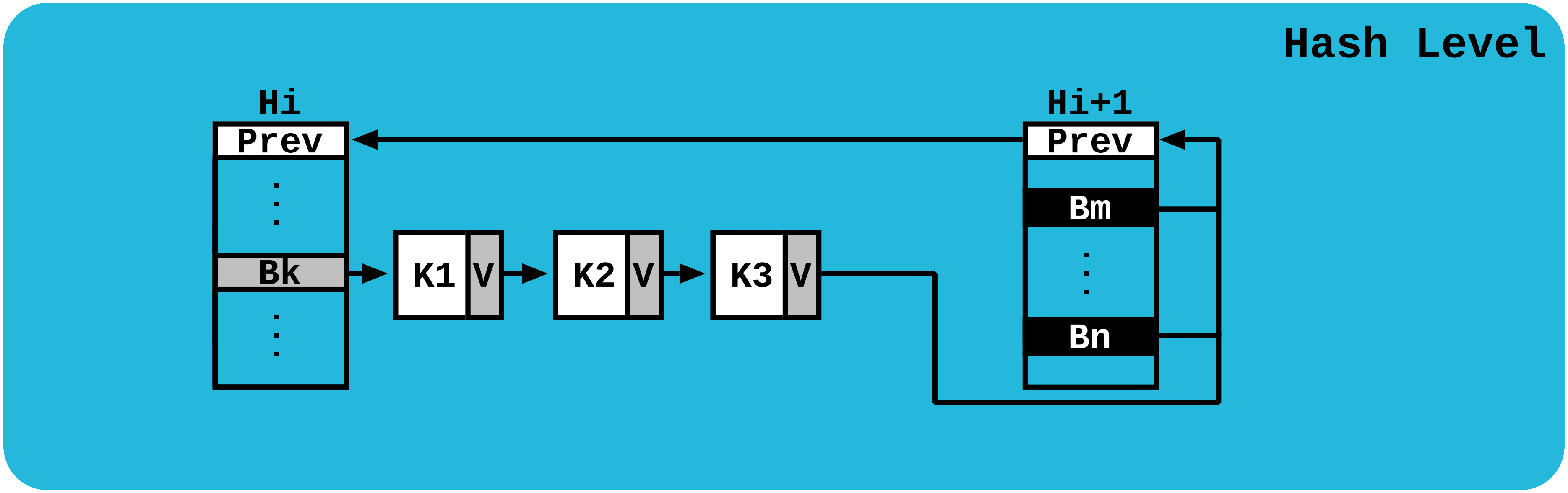


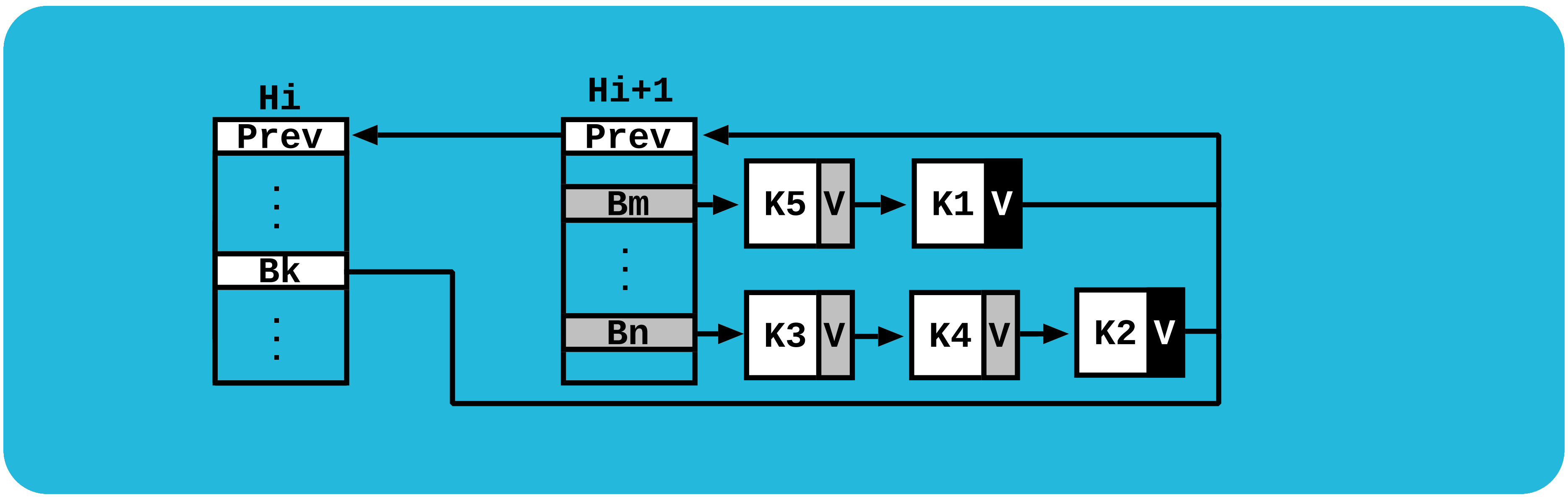
The LF Design





The LF Design





Performance Analysis

- **Hardware**: 32 (2 x 16) core **AMD** with 32 GB of main memory.
- **Software**: Linux **Fedora** 20 with **Oracle's Java Development Kit** 1.8.
- **Benchmarks**: Sets of 10^6 **randomized keys** with **insert**, **search** and **remove methods** (each **benchmark** had **5 warm up** runs and **20 standard** runs).
- **LF** design: **Expanded** with 6 **valid** nodes and had **two configurations** (**8 and 32 hash bucket** levels).

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Performance Analysis

- In the next slide, we will be comparing the **LF** design against other state-of-the-art **hash map** designs that support efficiently **concurrency**: Concurrent Hash Maps (**CH**), Concurrent Skip Lists (**CS**), Non Blocking Hash Maps (**NB**) and Concurrent Tries (**CT**).
- **Podium** colors: **first place**, **second place** and **third place**.

U.PORTO
FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Performance Analysis

➤ **Execution time** (**lower** is **better**) **Speedup Ratio** (**higher** is **better**).

# Threads ($\mathbf{T_p}$)	Execution Time (E_{T_p}) CH	CS	NB	CT	LF_8	LF_{32}	Speedup Ratio (E_{T_1}/E_{T_p}) CH	CS	NB	CT	LF_8	LF_{32}
1st – Insert: 60% Search: 30% Remove: 10%												
1	721	2,510	15,342	1,027	873	618						
8	150	413	4,030	174	148	142	4.81	6.08	3.81	5.90	5.90	4.35
16	128	247	2,803	115	91	106	5.63	10.16	5.47	8.93	9.59	5.83
24	75	191	2,566	89	72	74	9.61	13.14	5.98	11.54	12.13	8.35
32	72	178	1,870	90	80	67	10.01	14.10	8.20	11.41	10.91	9.22
2nd – Insert: 20% Search: 70% Remove: 10%												
1	282	1,890	12,370	764	757	395						
8	51	282	8,517	171	157	74	5.53	6.70	1.45	4.47	4.82	5.34
16	39	184	3,623	87	72	82	7.23	10.27	3.41	8.78	10.51	4.82
24	37	143	3,058	73	69	64	7.62	13.22	4.05	10.47	10.97	6.17
32	38	145	2,081	74	69	65	7.42	13.03	5.94	10.32	10.97	6.08
3th – Insert: 25% Search: 50% Remove: 25%												
1	279	2,059	12,181	1,087	808	440						
8	113	340	3,125	159	127	83	2.47	6.06	3.90	6.84	6.36	5.30
16	64	214	3,482	104	82	70	4.36	9.62	3.50	10.45	9.85	6.29
24	42	180	2,609	87	71	78	6.64	11.44	4.67	12.49	11.38	5.64
32	44	166	1,902	83	77	66	6.34	12.40	6.40	13.10	10.49	6.67

Questions & (Possible) Answers

- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)

Questions & (Possible) Answers

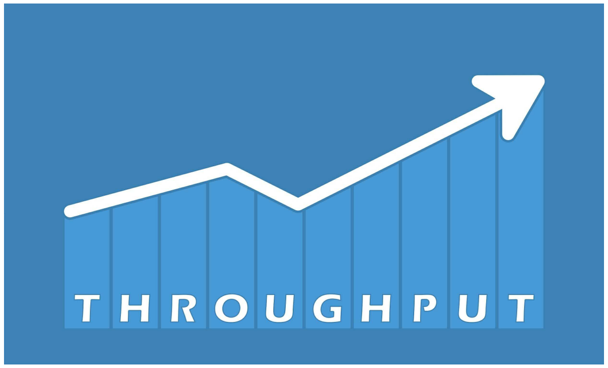
- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
  - Depending on the **kind of the problem**, one can use a **concurrent** approach to solve it faster.

Questions & (Possible) Answers

- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
  - Depending on the **kind of the problem**, one can use a **concurrent** approach to solve it faster.
  - **Lock-freedom** can be consider to be **fast**. **Individual threads** do tend to **execute more work**, but in face of **heavy contention**, **lock-freedom** is known to **improve dramatically** the overall **throughput**.

Questions & (Possible) Answers

- What is the **best way** to run the process? What is the **definition** of **best way**? (fastest, resource efficient, tolerant to problems, ...)
  - Depending on the **kind of the problem**, one can use a **concurrent** approach to solve it faster.
  - **Lock-freedom** can be consider to be **fast**. **Individual threads** do tend to **execute more work**, but in face of **heavy contention**, **lock-freedom** is known to **improve dramatically** the overall **throughput**.



  - And, as shown before, **lock-freedom** is also known to **avoid important problems** related with **concurrency** (deadlocks, convoying, kill-tolerant...)

Questions & (Possible) Answers

- In this process, what can we **control**? Can we **control** the **environment**? (e.g. can we control the police?)

Questions & (Possible) Answers

- In this process, what can we **control**? Can we **control** the **environment**? (e.g. can we control the police?)
  - **Lock-freedom** cannot control the environment (nor the police), but it makes the **progress** of a **concurrent computation independent** of it (periodical table).

U.PORTO
FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Questions & (Possible) Answers

- How should we implement the **flows**? Having **multiple flows** should be better, right?
- Well...It depends. **Critical regions** are always a **problem** (e.g. crossroads).

Questions & (Possible) Answers

- How should we implement the **flows**? Having **multiple flows** should be better, right?
- Well...It depends. **Critical regions** are always a **problem** (e.g. crossroads).
  - **Lock-freedom** does not avoid critical regions, but by **minimizing their size** (atomic instructions), we can argue that we can understand better the behavior of a concurrent computation.

Thank You !!!

Special Thanks: *Ricardo Rocha (insightful suggestions)*
Rita Ribeiro (logistics)
Sérgio Crisóstomo (invitation)

LF design: ***https://github.com/miar/ffps***
FCT grant: *SFRH/BPD/108018/2015*

TALKS @ [dcc]