QPipe: Quantiles Sketch Fully in the Data Plane

Zhuolong Yu

Johns Hopkins University

with Nikita Ivkin, Vladimir Braverman, Xin Jin

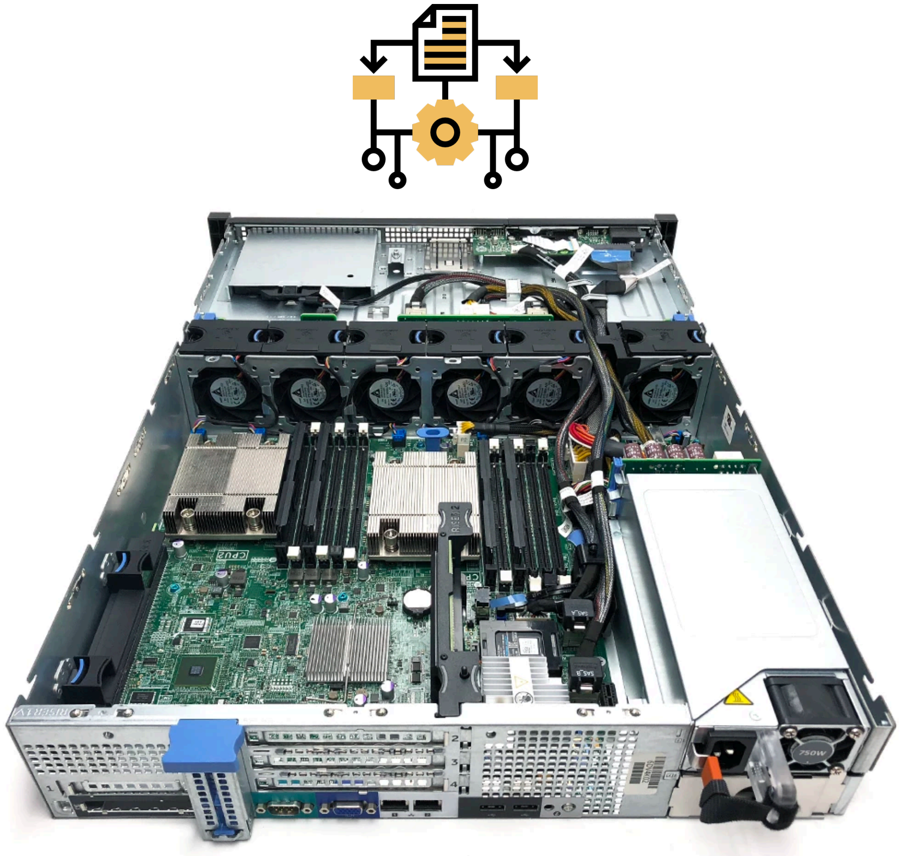
CoNEXT 2019

Statistics over packet flows

Efficient network management requires a variety of statistics

Statistics over packet flows

Efficient network management requires a variety of statistics



Server running measurement job

Traffic Engineering

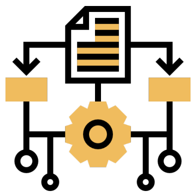
Heavy hitter detection

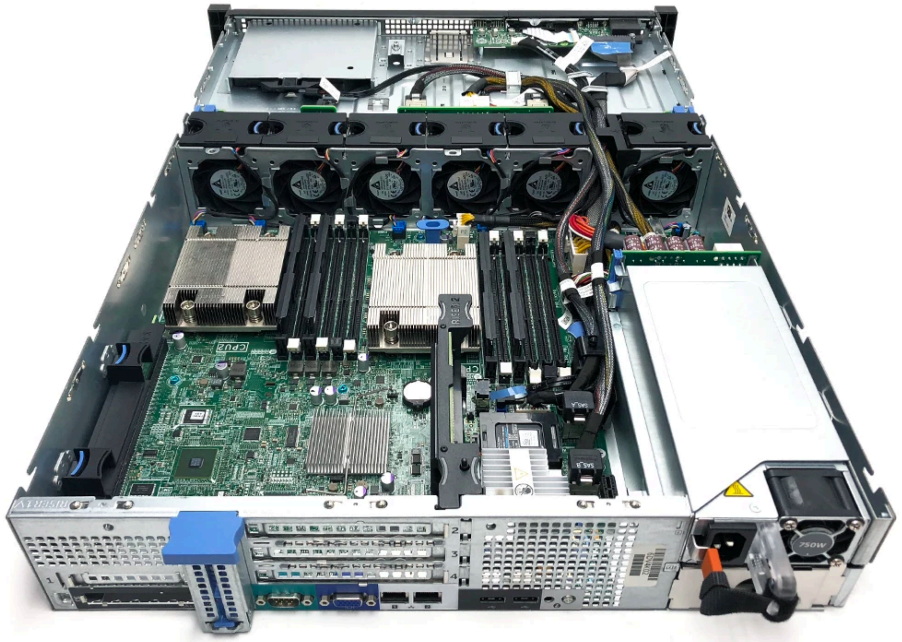
Worm Detection

Accounting

Statistics over packet flows

Programmable switch enriches the operations on the data plane



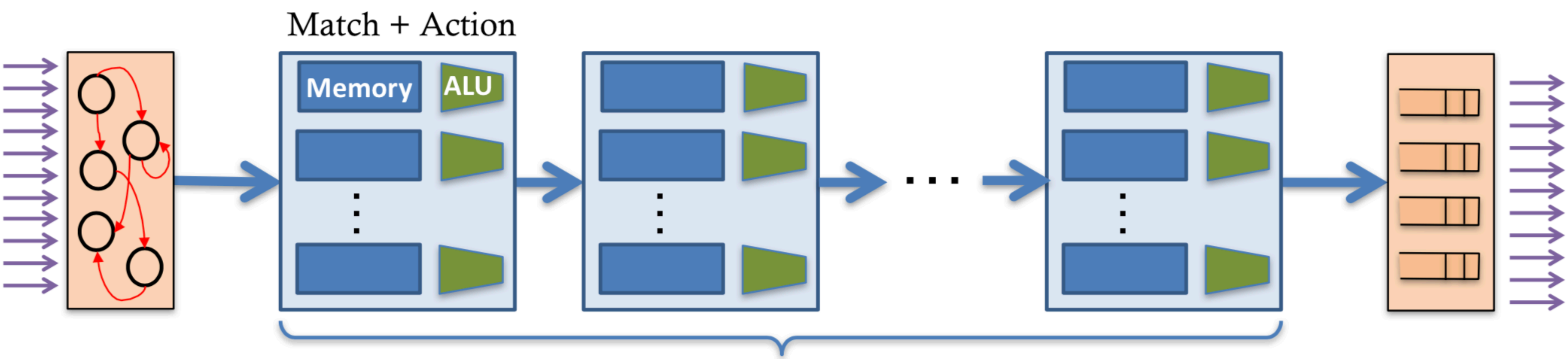


High packet processing rate!

Statistics over packet flows

Programmable switch enriches the operations on the data plane

PISA: Protocol Independent Switch Architecture



Programmable Parser

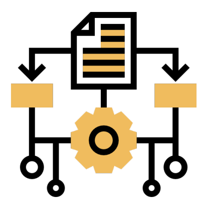
Programmable Match-Action Pipeline

Converts packet data into metadata

Operate on metadata and update memory states

Statistics over packet flows

Programmable switch enriches the operations
on the data plane

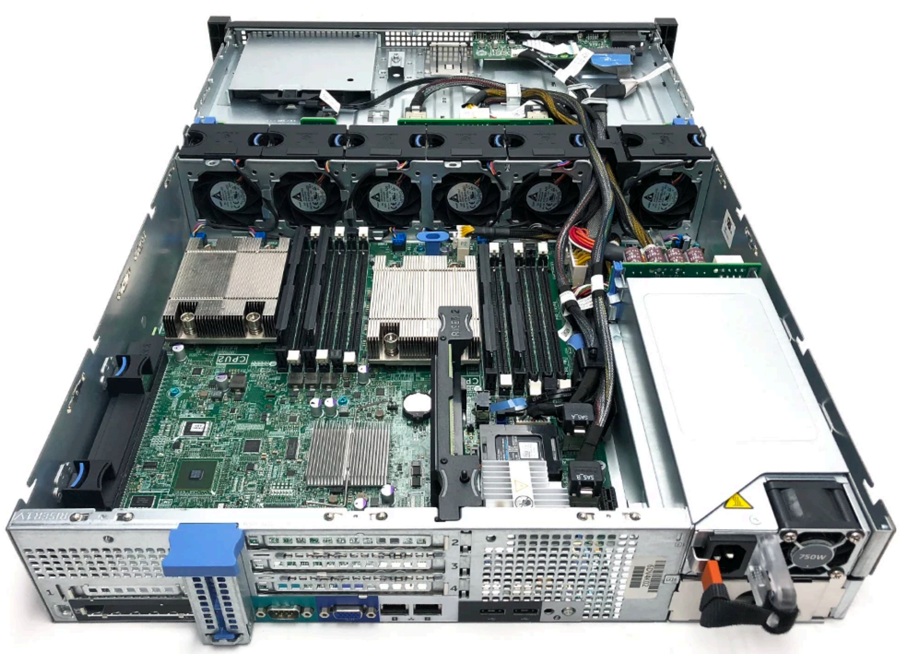


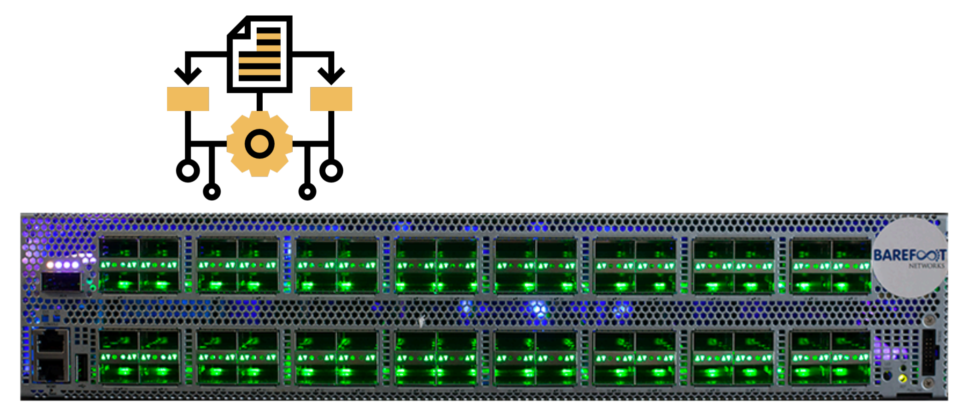
High packet processing rate!

Statistics over packet flows

Programmable switch enriches the operations on the data plane

Run measurement directly in data plane!





High packet processing rate!

Statistics over packet flows

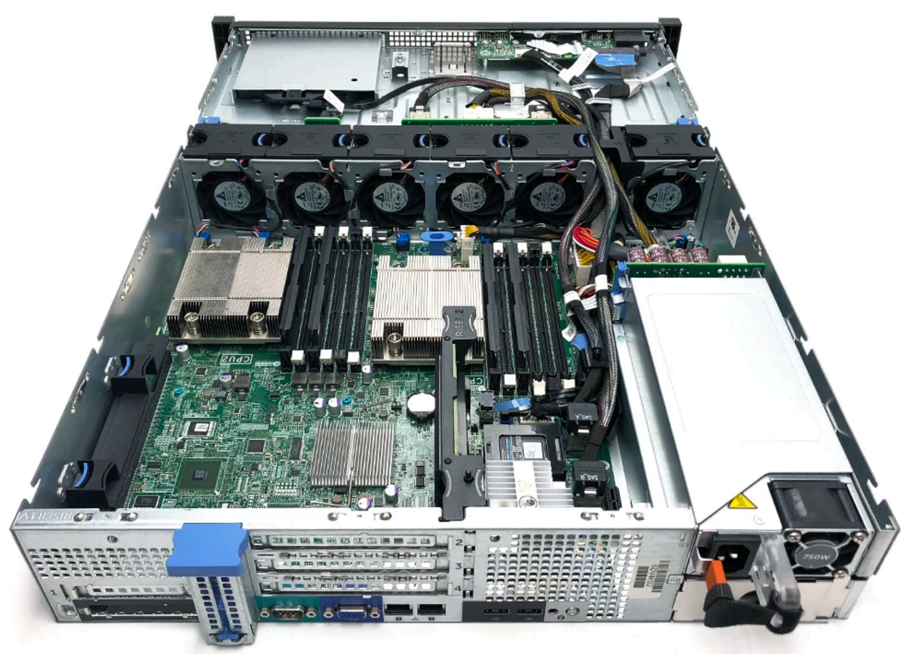
Programmable switch enriches the operations
on the data plane

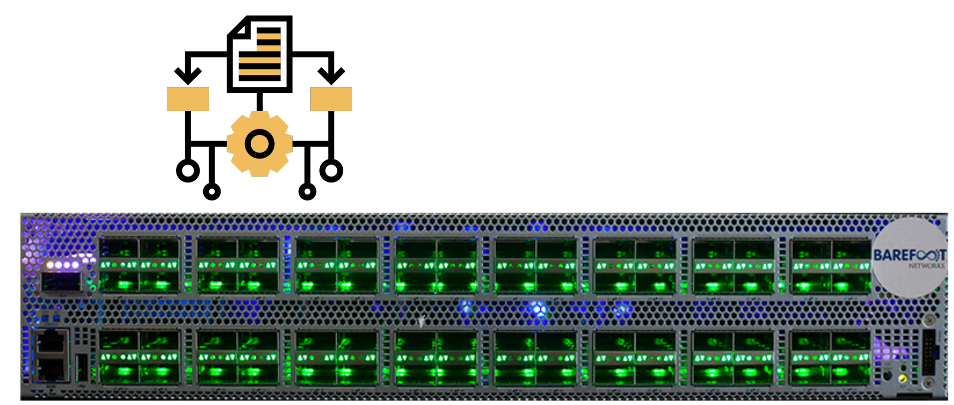
Run measurement directly in data plane!

CM-Sketch

UnivMon-sigcomm16

Hashpipe-sosr17





High packet processing rate!

Statistics over packet flows

Programmable switch enriches the operations
on the data plane

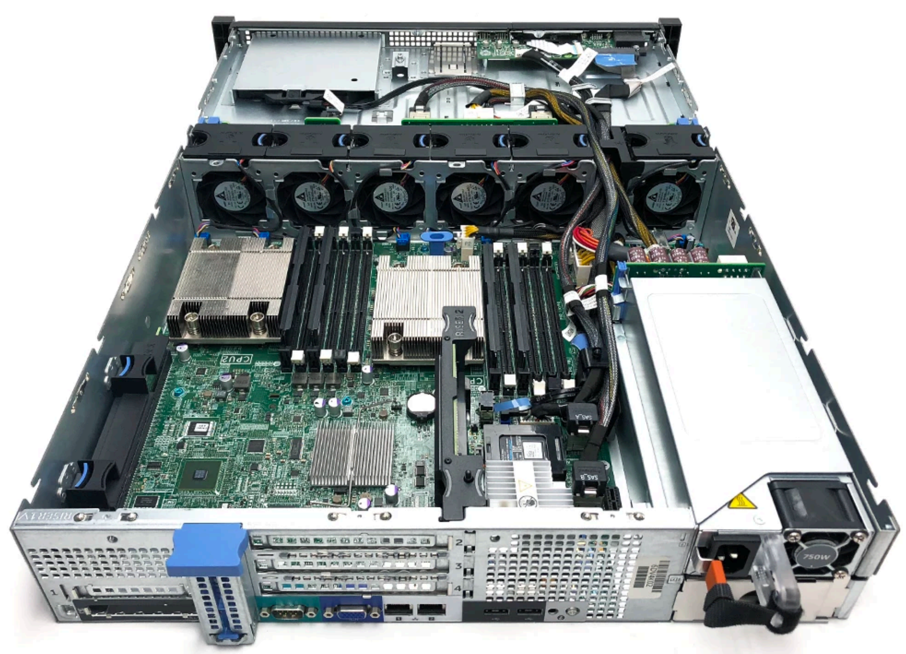
Run measurement directly in data plane!

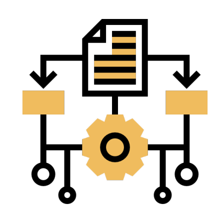
CM-Sketch

UnivMon-sigcomm16

Hashpipe-sosr17

Finding Quantile?





High packet processing rate!

Statistics over packet flows

Programmable switch enriches the operations on the data plane

Given stream $S=s_1,\ldots,s_n$

(1) For query x, return the rank r(x), i.e., number of items smaller than x in S.

(2) For rank query i, return i-th smallest item.

n data plane!

CM-Sketch

UnivMon-sigcomm16

Hashpipe-sosr17

Finding Quantile?

High packet processing rate!

Statistics over packet flows

A simple way to report quantile is **Packet Sampling**

Statistics over packet flows

A simple way to report quantile is **Packet Sampling**

Requires **large memory** to achieve certain accuracy if the flow stream is large

Statistics over packet flows

A simple way to report quantile is **Packet Sampling**

Requires **large memory** to achieve certain accuracy if the flow stream is large

Switch ASICs only have **tens of MBs** of memory!

Statistics over packet flows

A simple way to report quantile is **Packet Sampling**

Requires **large memory** to achieve certain accuracy if the flow stream is large

Switch ASICs only have **tens of MBs** of memory!

A memory efficient way to do sampling: **KLL**

Statistics over packet flows

A simple way to report quantile is **Packet Sampling**

Requires **large memory** to achieve certain accuracy if the flow stream is large

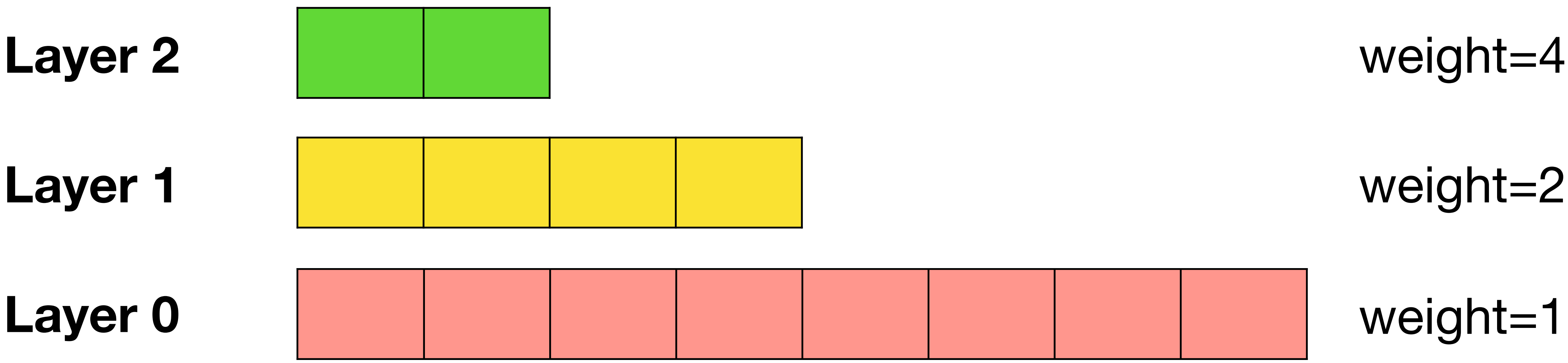
Switch ASICs only have **tens of MBs** of memory!

A memory efficient way to do sampling: **KLL**

Zohar Karnin, Kevin Lang, and Edo Liberty, FOCS 2016

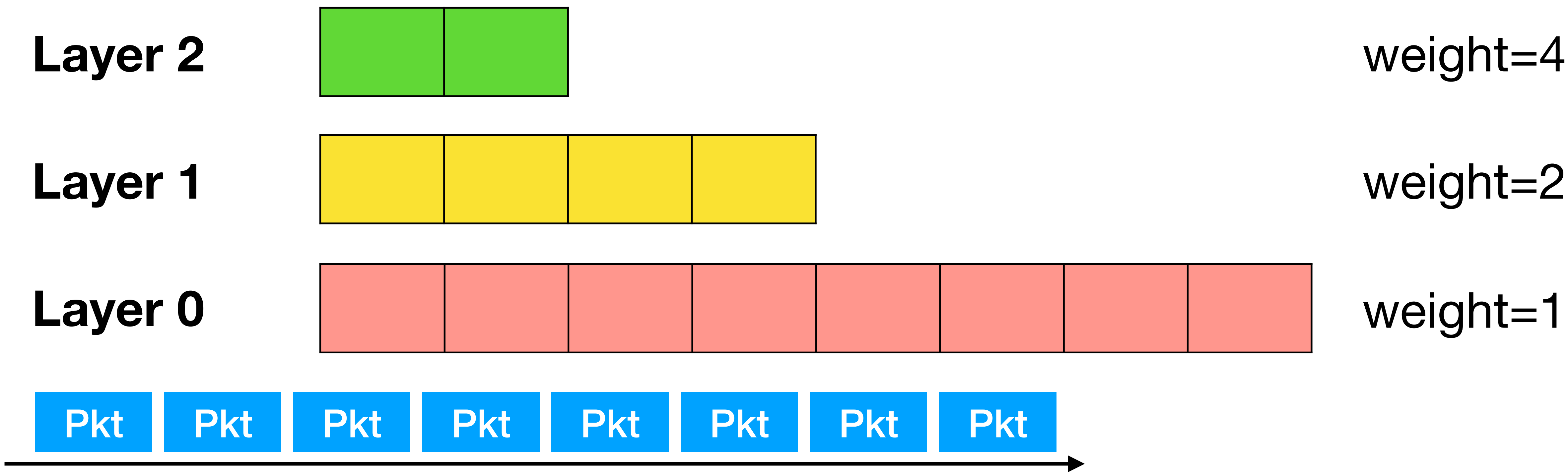
Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



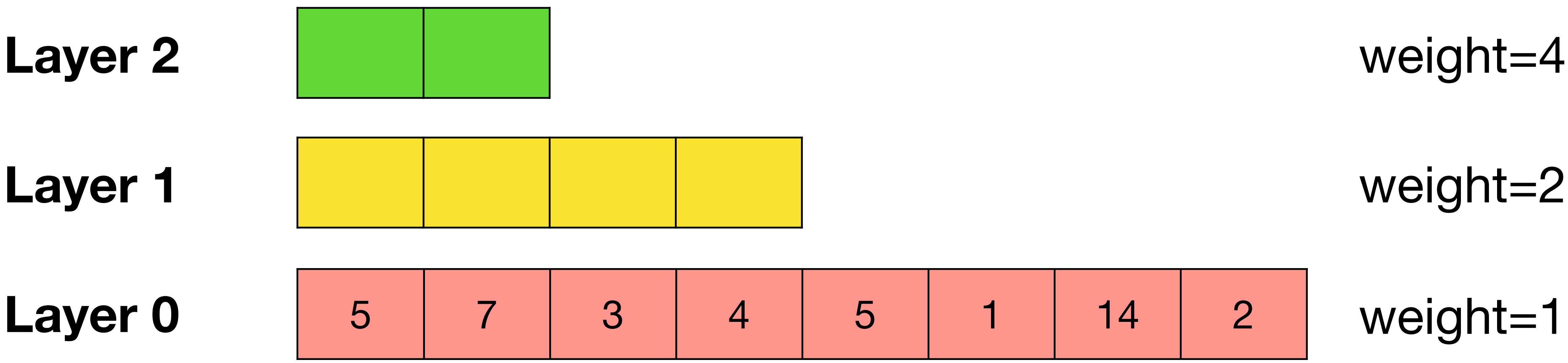
Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



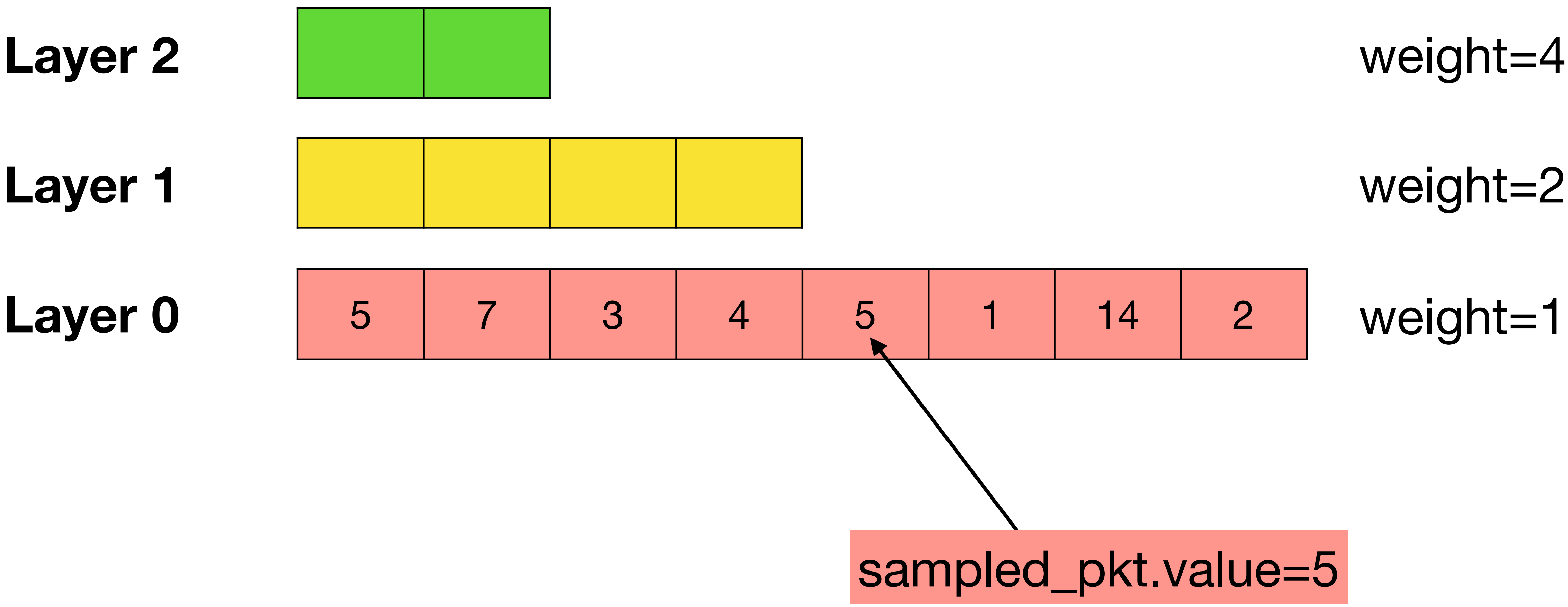
Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



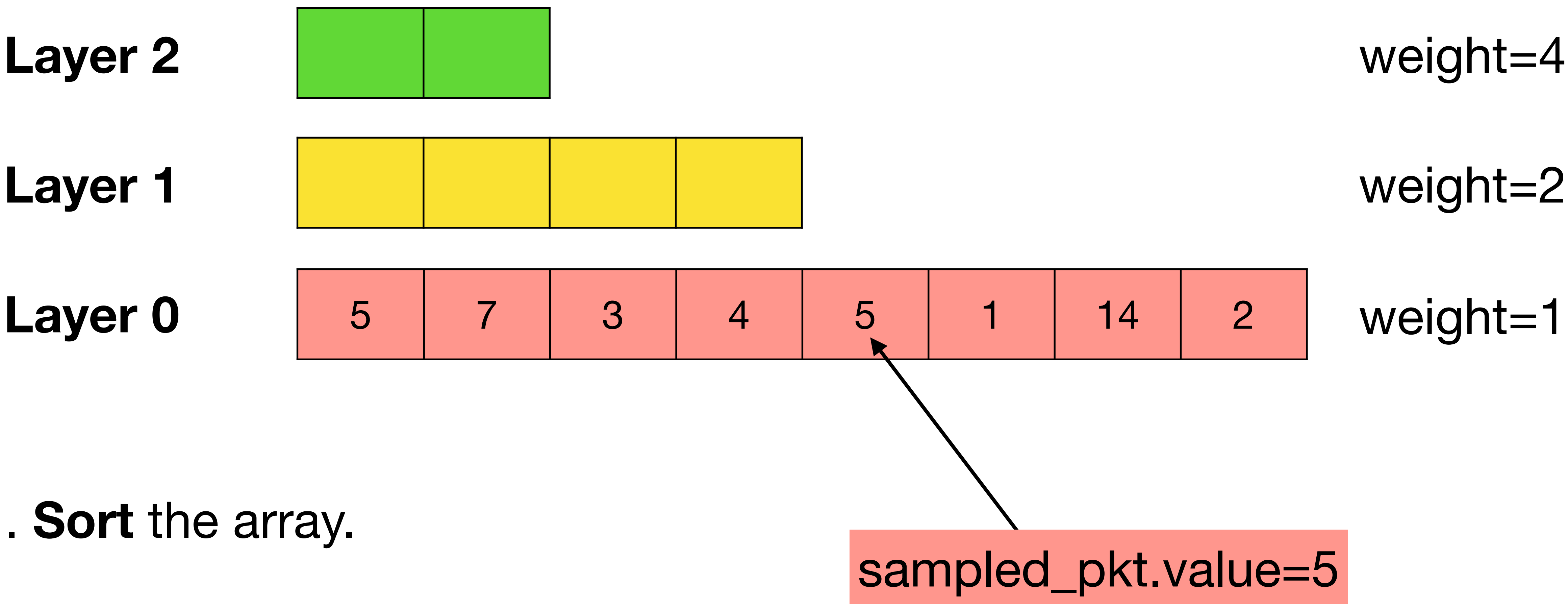
Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



Efficient quantile streaming

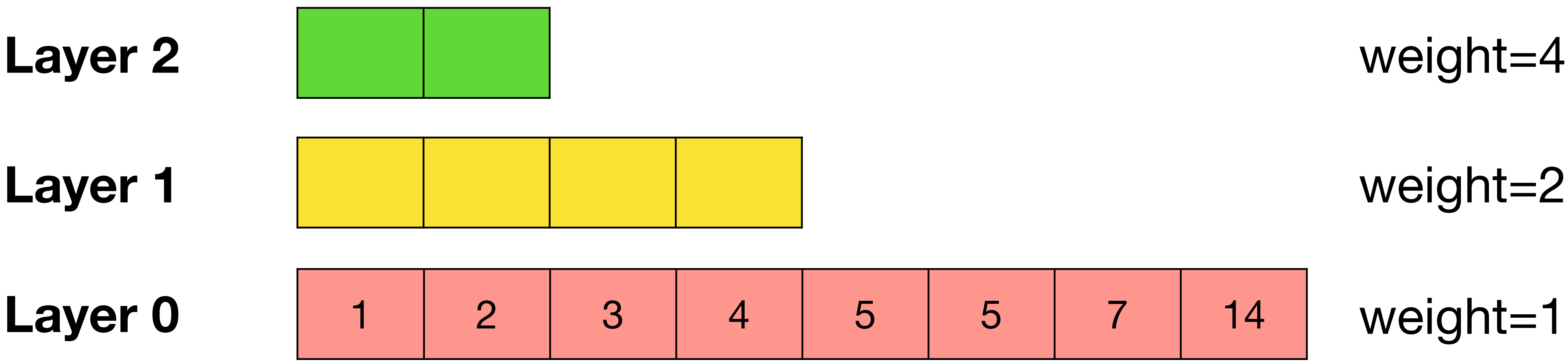
Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



1. **Sort** the array.

Efficient quantile streaming

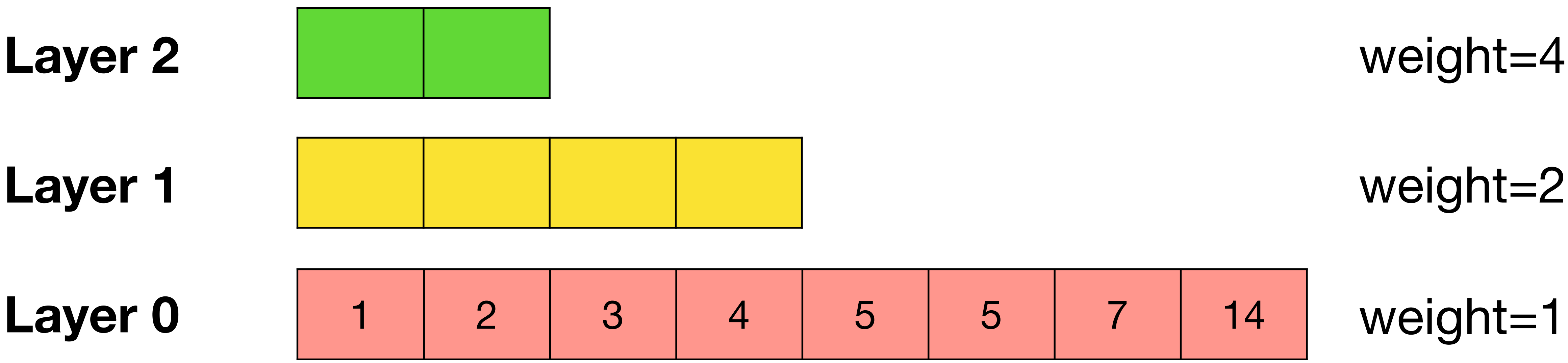
Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



1. **Sort** the array.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

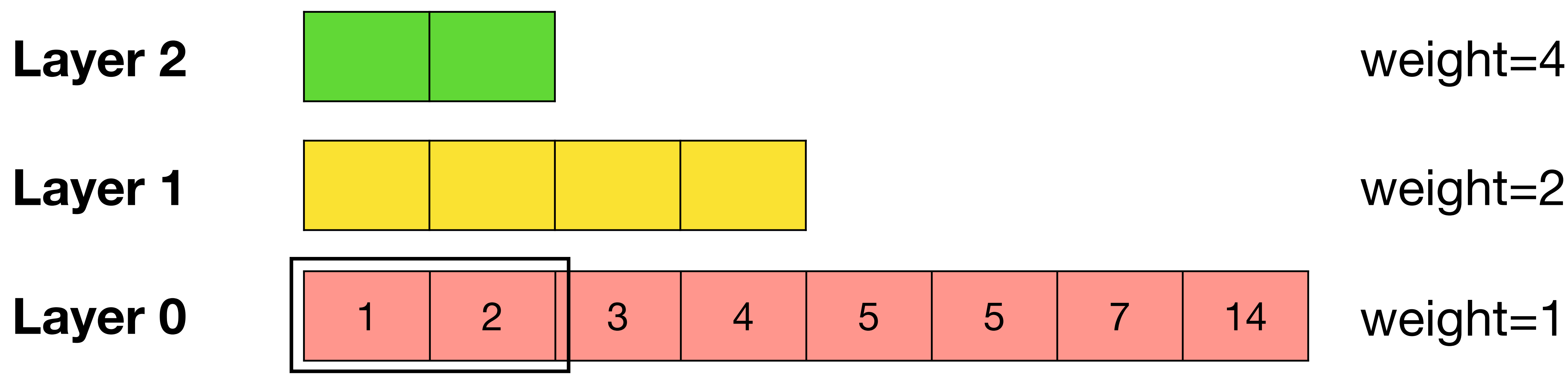


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

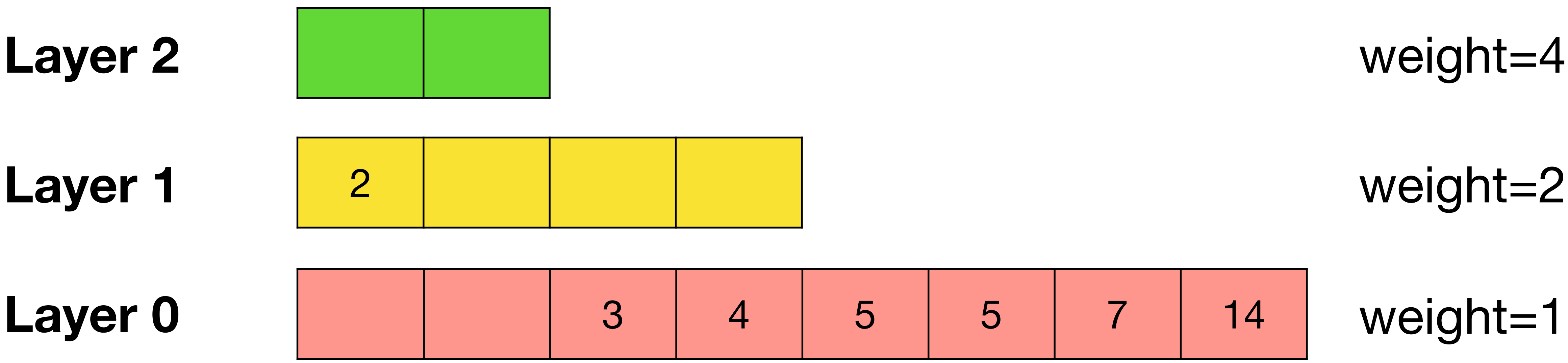


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

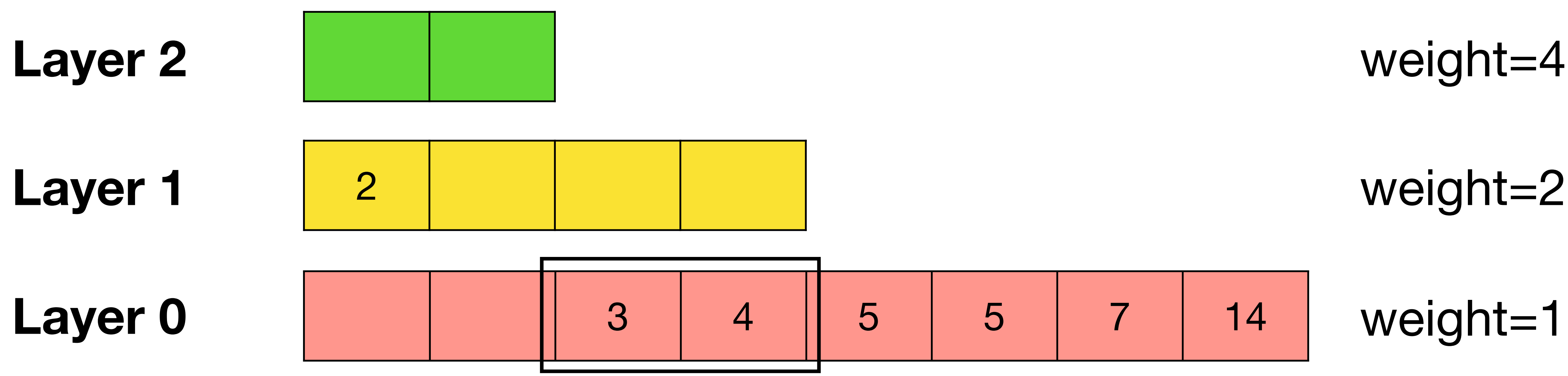


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

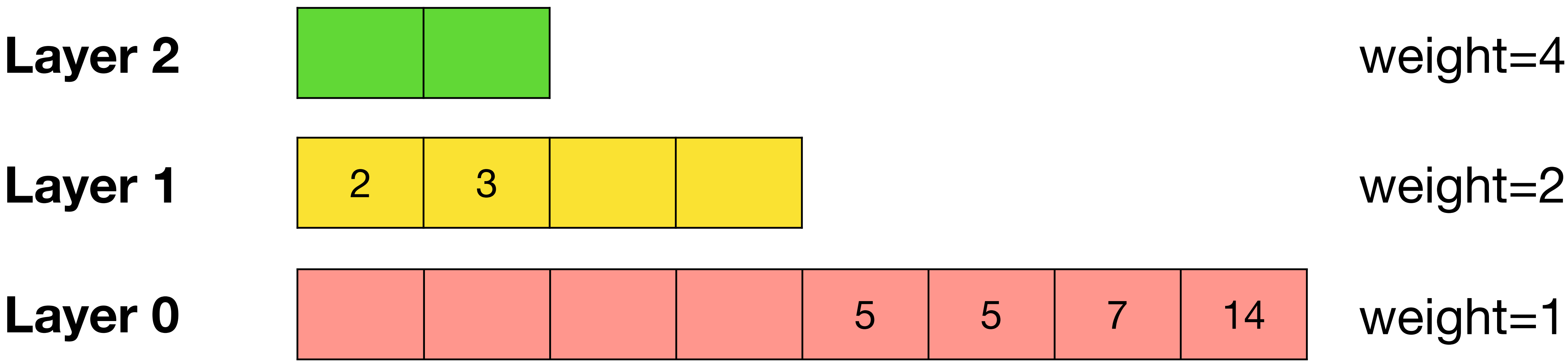


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

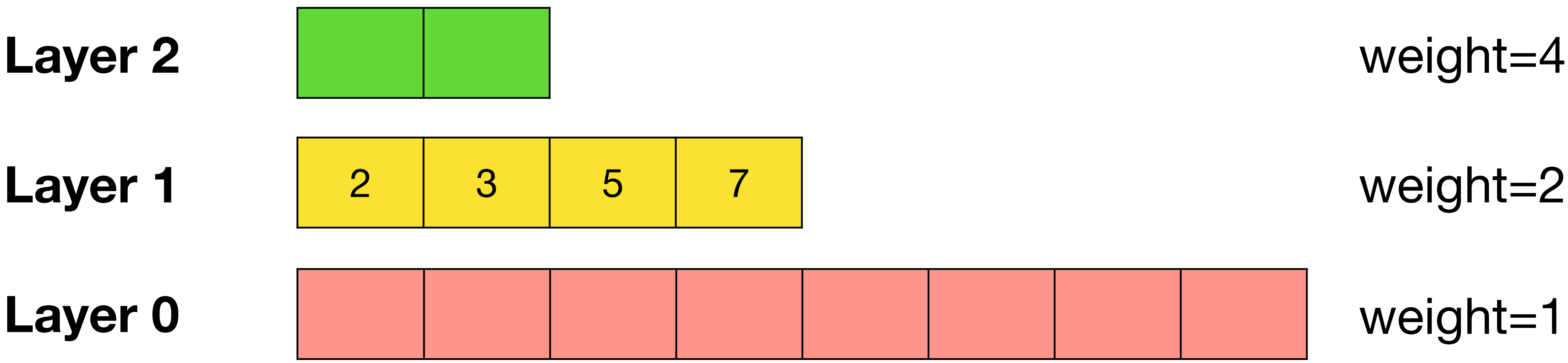


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.

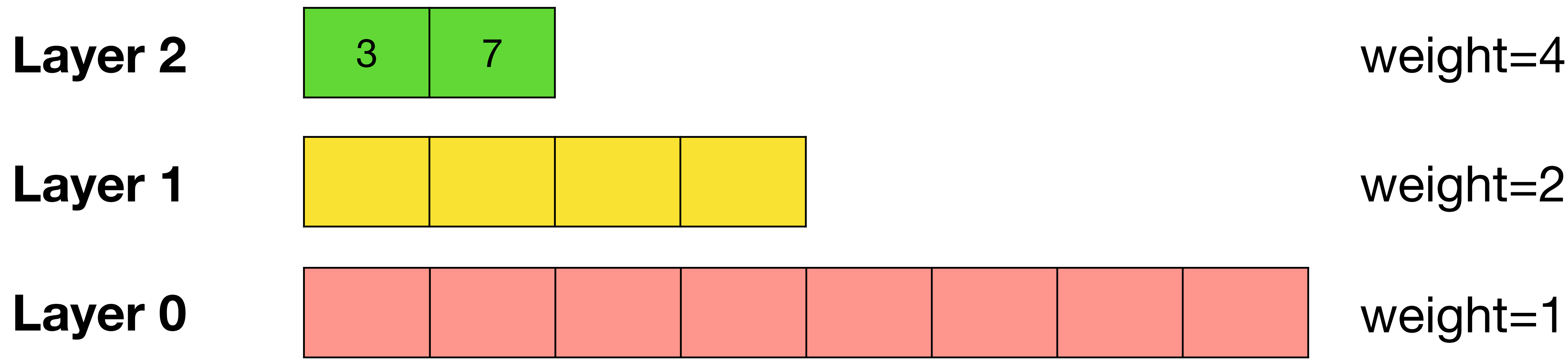


1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

Efficient quantile streaming

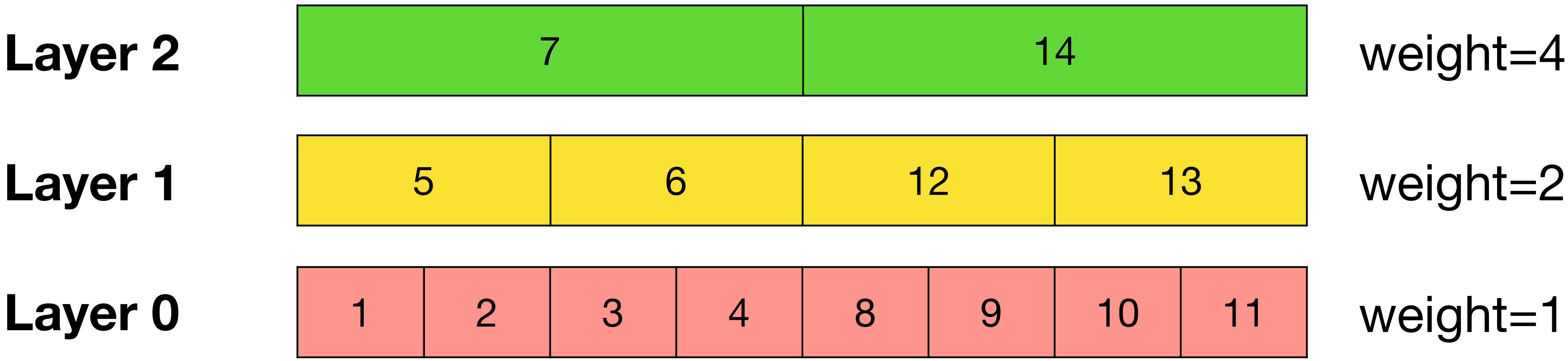
Instead of sampling packets into a flat array, KLL stores them in a hierarchical way.



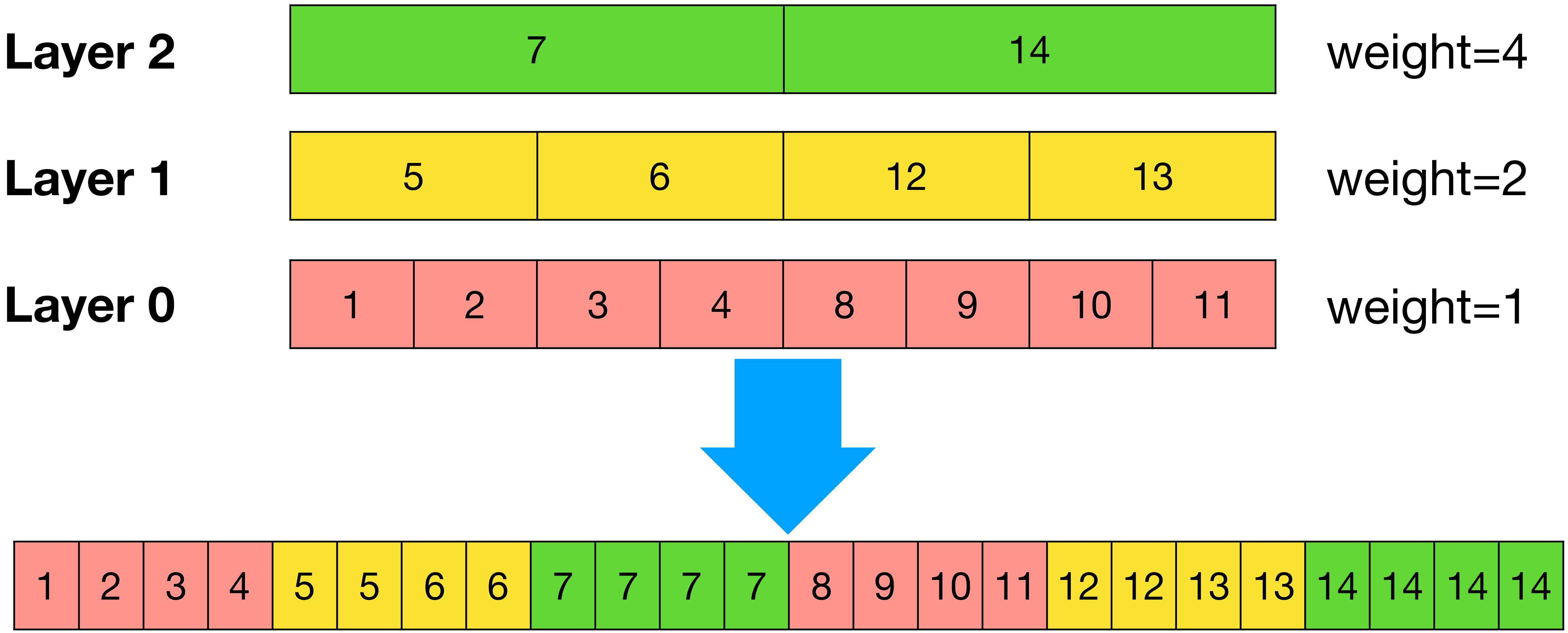
1. **Sort** the array.

2. **Subsample**: go through the array in order, randomly feed one item to the next layer and drop the other item.

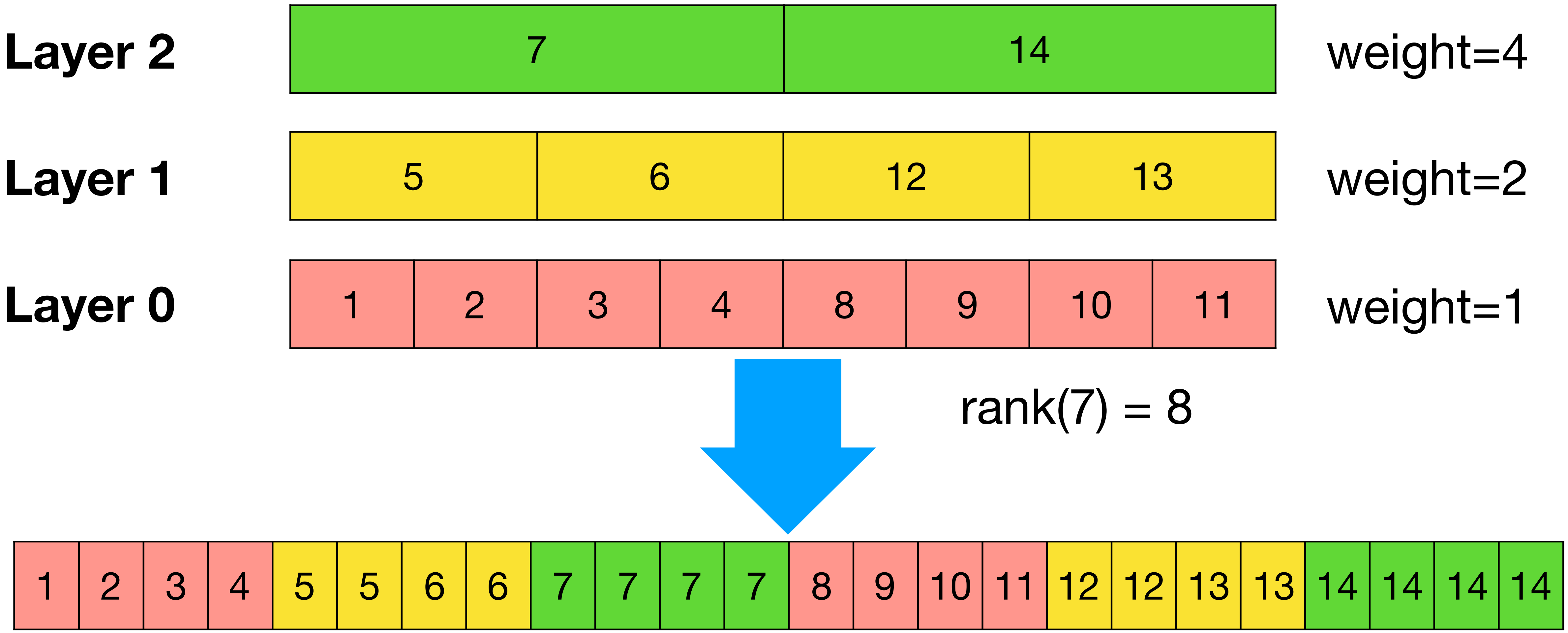
Efficient quantile streaming



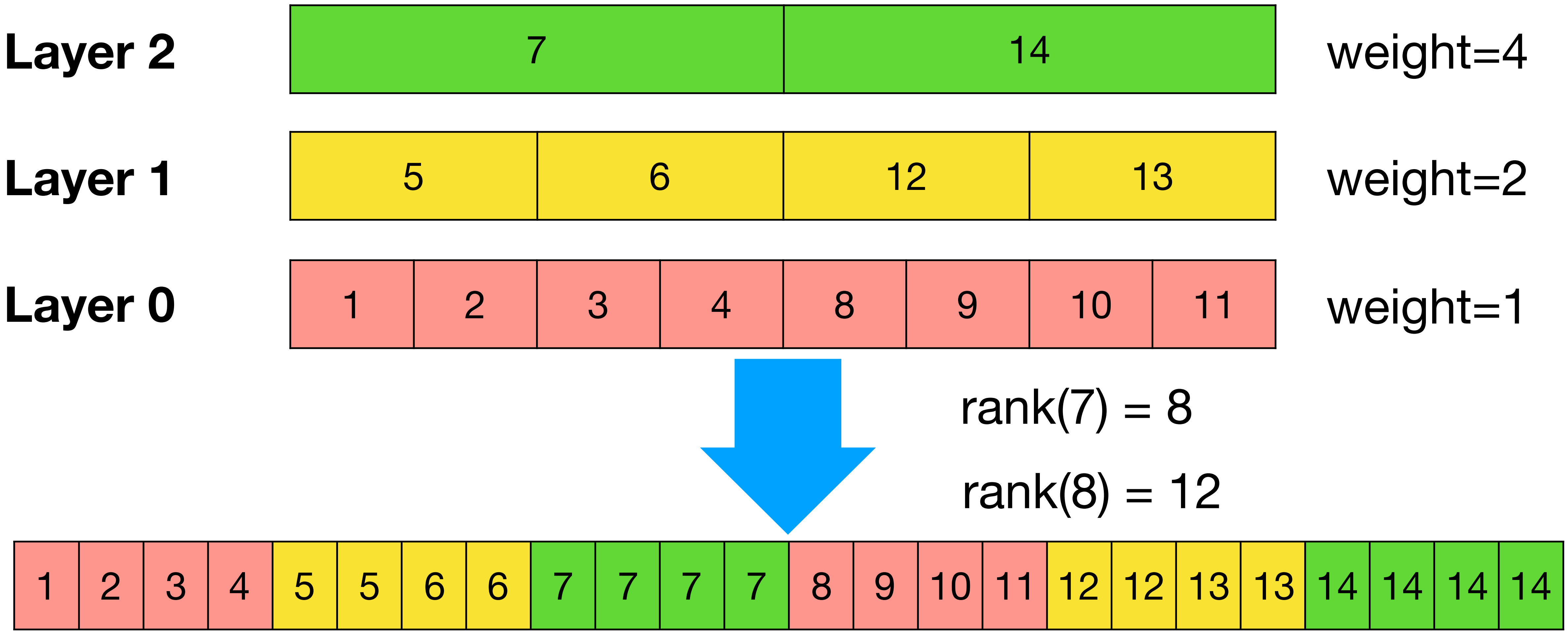
Efficient quantile streaming



Efficient quantile streaming



Efficient quantile streaming



Efficient quantile streaming

Layer 2	7	14	weight=4
Layer 1	5, 6	12, 13	weight=2
Layer 0	1, 2, 3, 4	8, 9, 10, 11	weight=1

rank(7) = 8

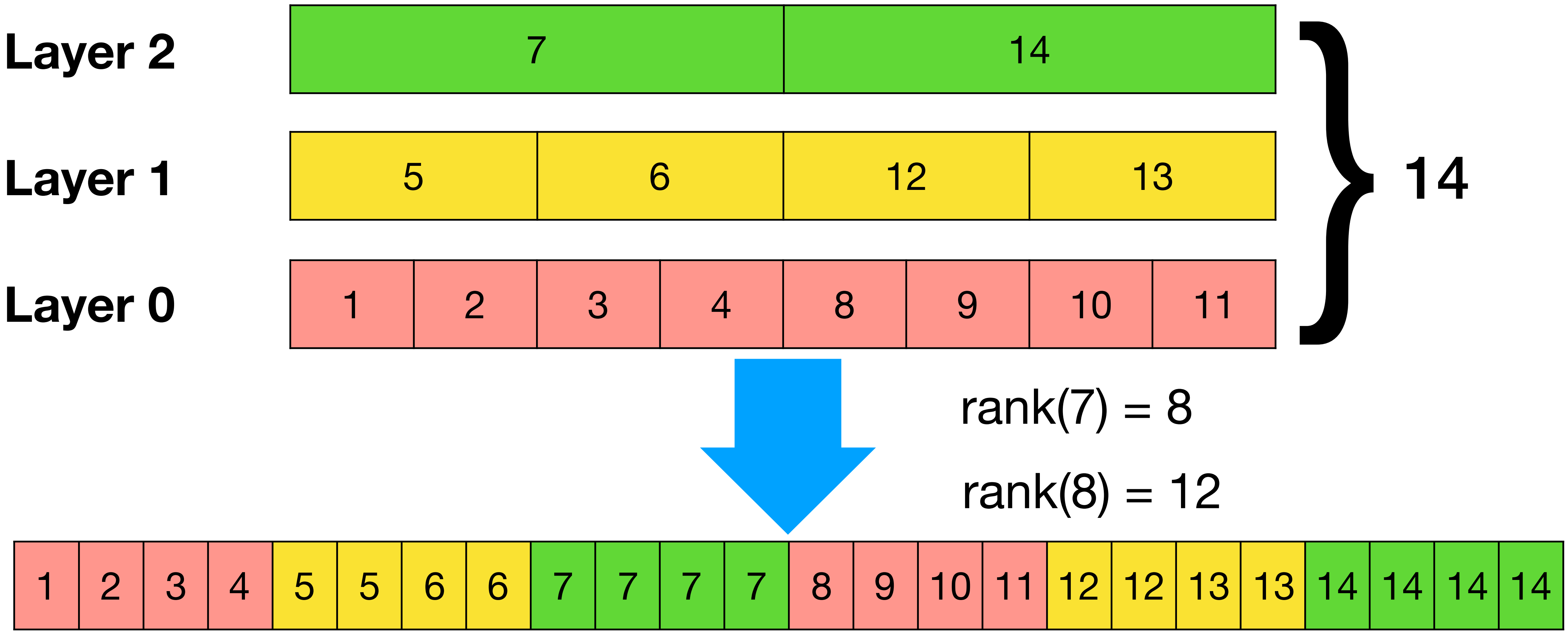
rank(8) = 12

1	2	3	4	5	5	6	6	7	7	7	7	8	9	10	11	12	12	13	13	14	14	14	14

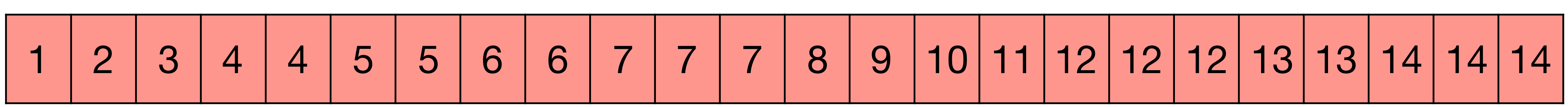
Basic sampling:

1	2	3	4	4	5	5	6	6	7	7	7	8	9	10	11	12	12	12	13	13	14	14	14

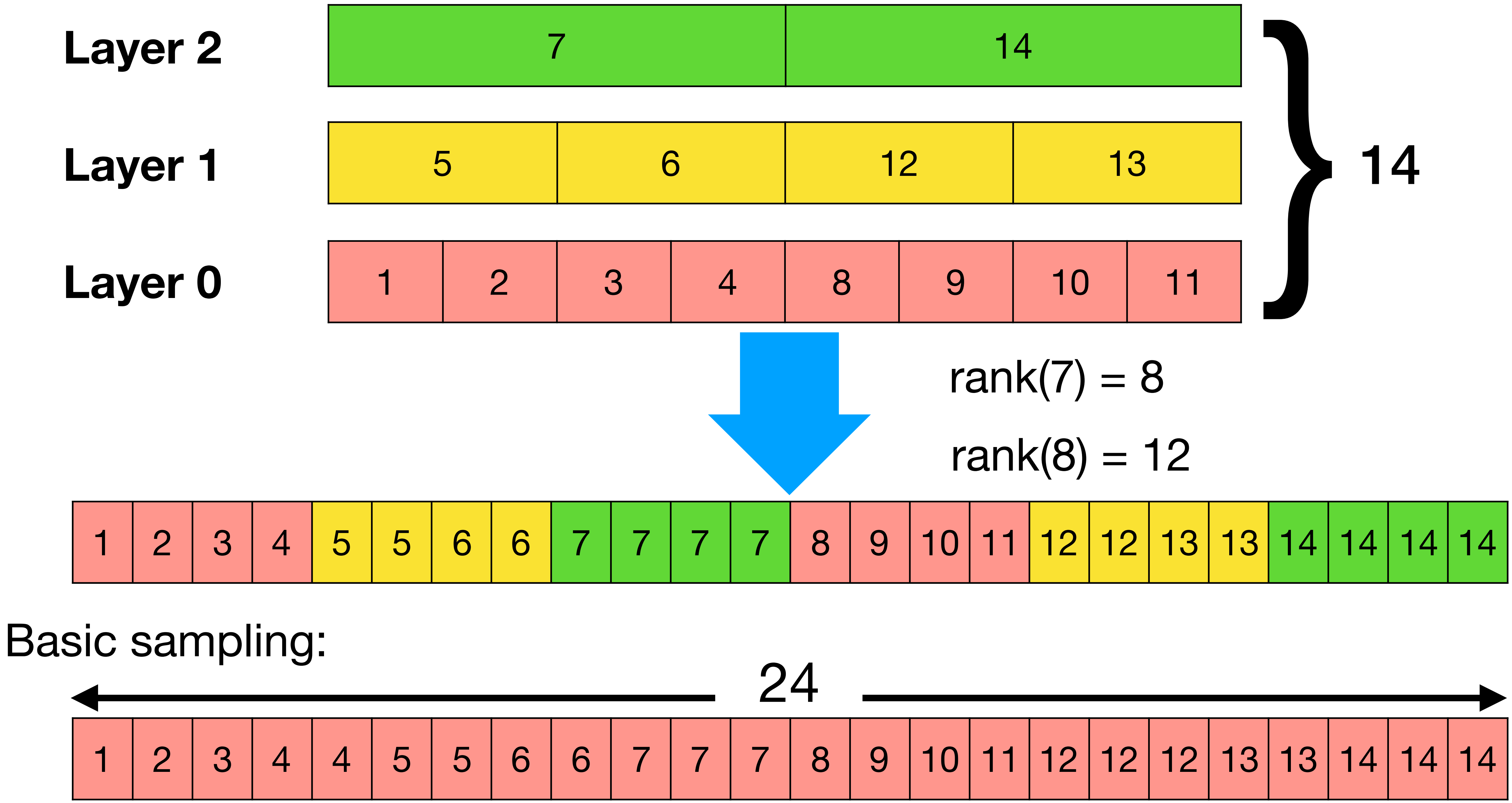
Efficient quantile streaming



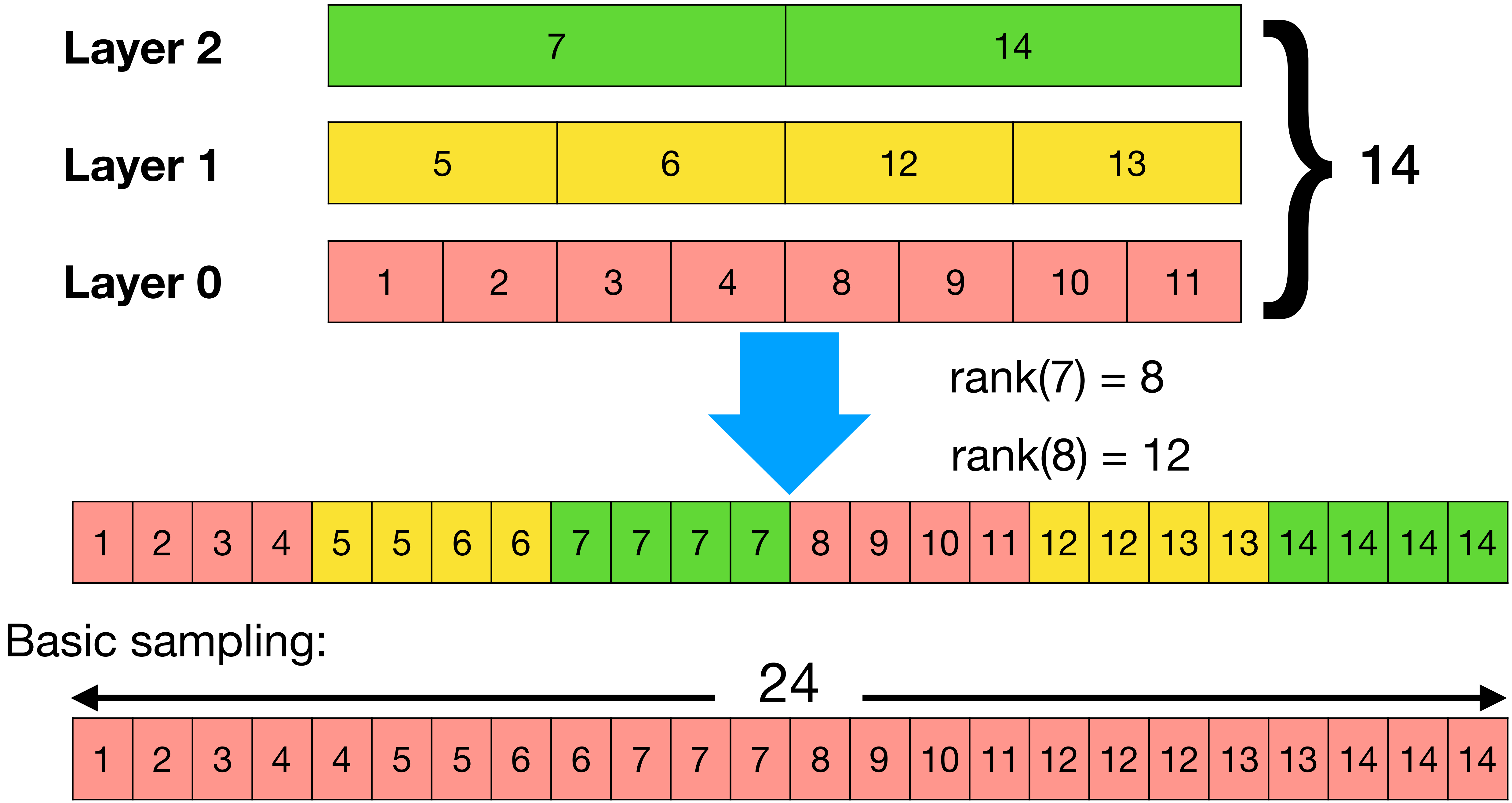
Basic sampling:



Efficient quantile streaming



Efficient quantile streaming



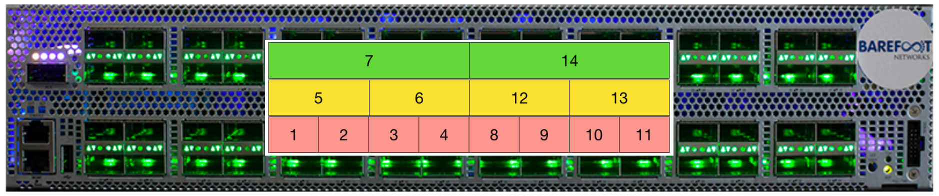
Guarantee: KLL preserves ranks with approximation $\pm\epsilon n$, given the memory budget of $O(\frac{1}{\epsilon}\log\log\frac{1}{\epsilon})$.

While basic sampling requires $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$.

System Design

<table>
<tr><td colspan="4">7</td><td colspan="4">14</td></tr>
<tr><td colspan="2">5</td><td colspan="2">6</td><td colspan="2">12</td><td colspan="2">13</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
</table>

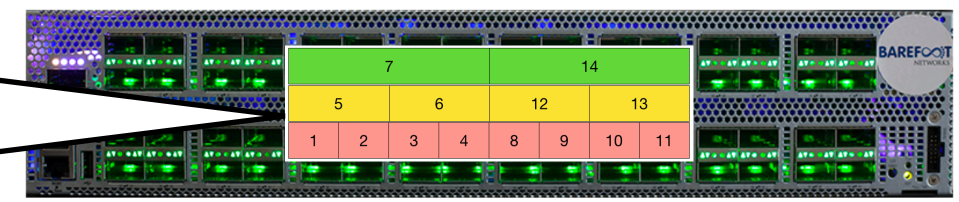
System Design



System Design

I need do sorting.

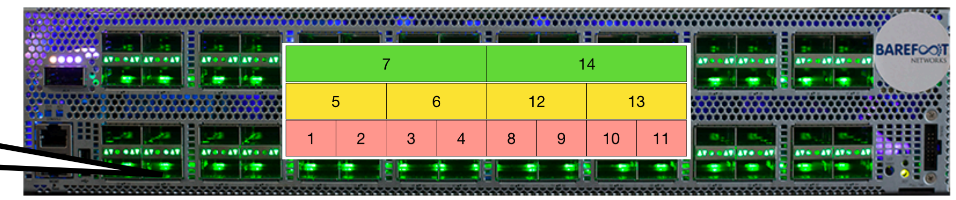
sort()



System Design

No, you can't.

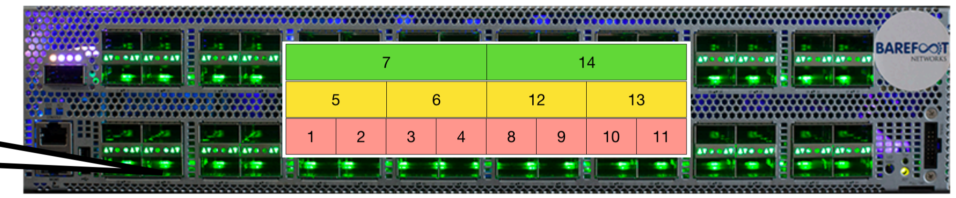
~~sort()~~



System Design

No, you can’t.

~~sort()~~

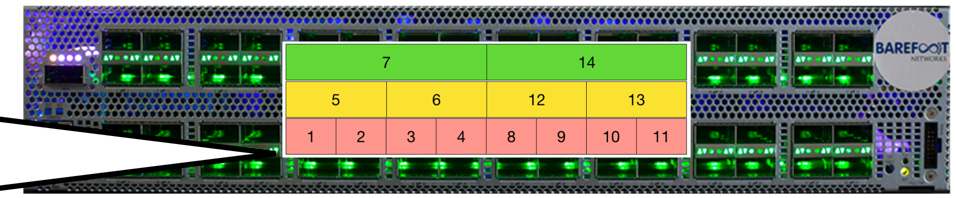


Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



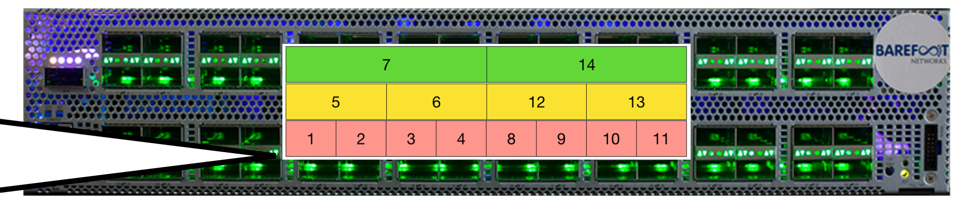
Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

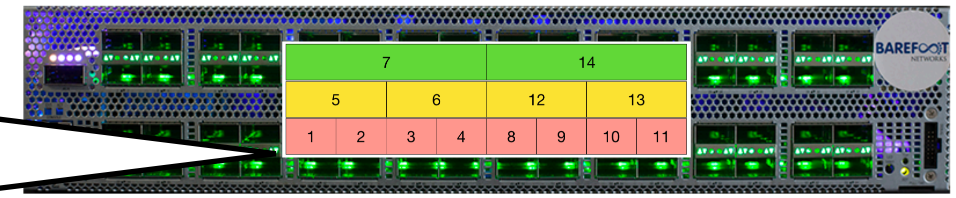
theta=0

5	7	3	4	5	1	14	2

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta

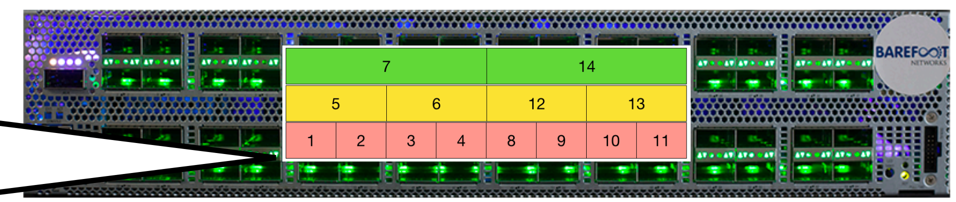
theta=0

5	7	3	4	5	1	14	2

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta
2. Subsample them

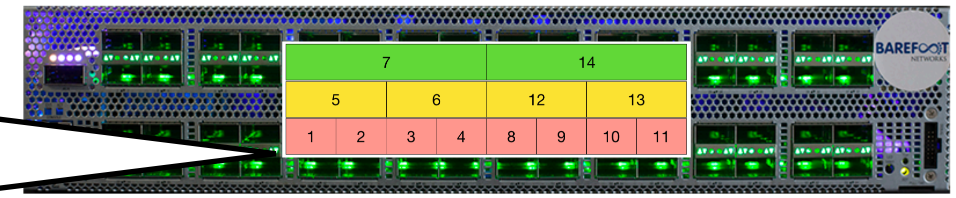
theta=0

5	7	3	4	5		14	

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta
2. Subsample them
3. Update theta as the larger subsampled item

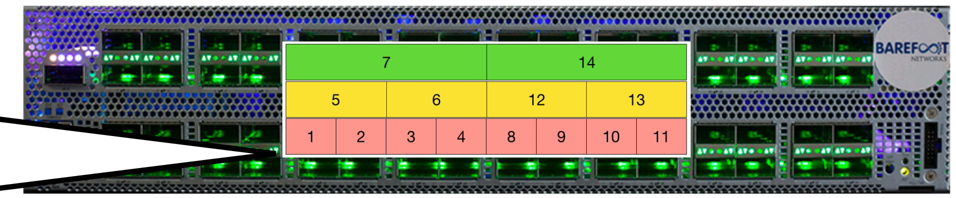
theta=2

5	7	3	4	5		14	

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta
2. Subsample them
3. Update theta as the larger subsampled item

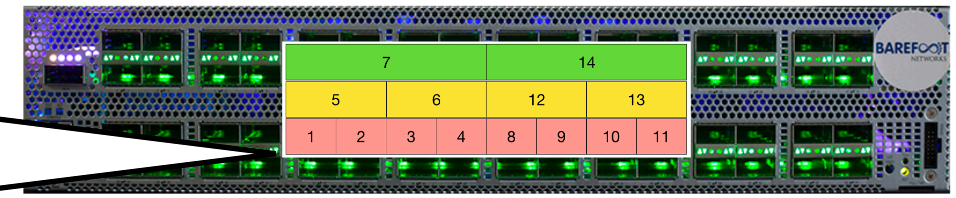
theta=2

5	7	3	4	5		14	

System Design

~~sort()~~ argmin()

Okay, argmin() maybe



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta
2. Subsample them
3. Update theta as the larger subsampled item

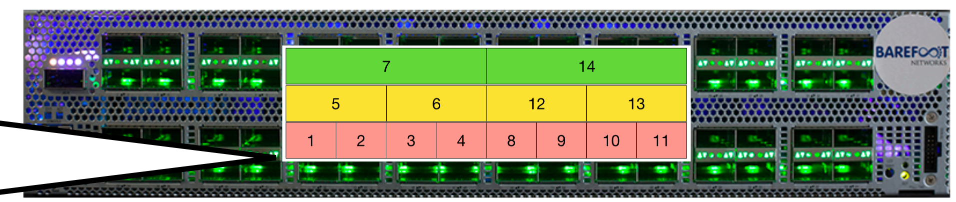
theta=4

5	7			5		14	

System Design

Okay, argmin() maybe

~~sort()~~ argmin()



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

1. Find two minimum items larger than theta
2. Subsample them
3. Update theta as the larger subsampled item

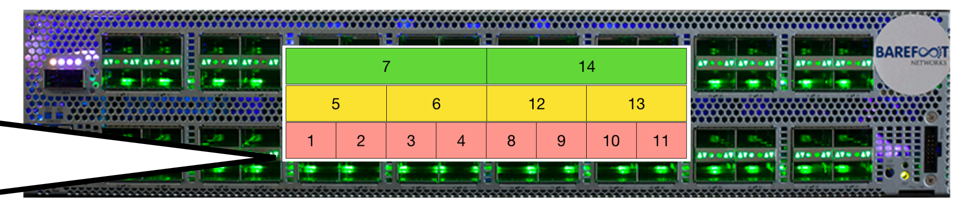
theta=5

	7					14	

System Design

~~sort()~~ argmin()

Okay, argmin() maybe



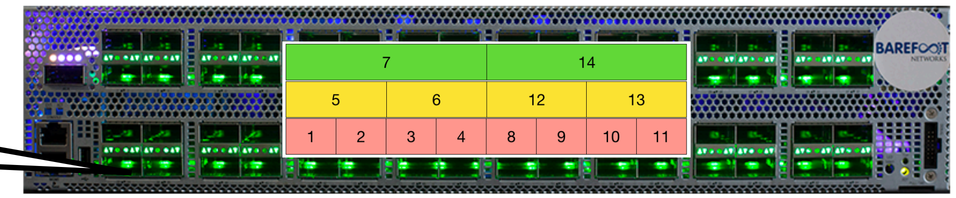
Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

System Design

No, you can't.

~~sort()~~ ~~argmin()~~



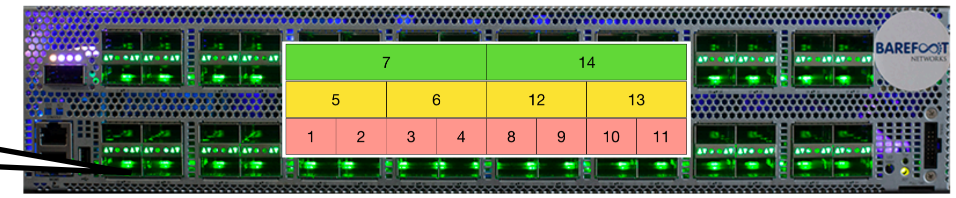
Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

System Design

No, you can't.

~~sort()~~ ~~argmin()~~



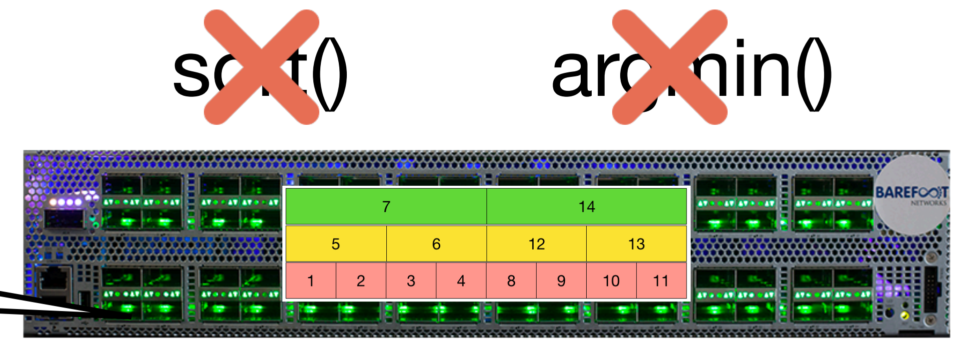
Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

We can use **argmin()** to find the two minimum items and subsample them

Observation: Large portion of **unsampled packets** go through switch pipeline anyway

System Design

No, you can't.



Challenge: Programmable switches only support simple operations (read/write/simple arithmetic logic)

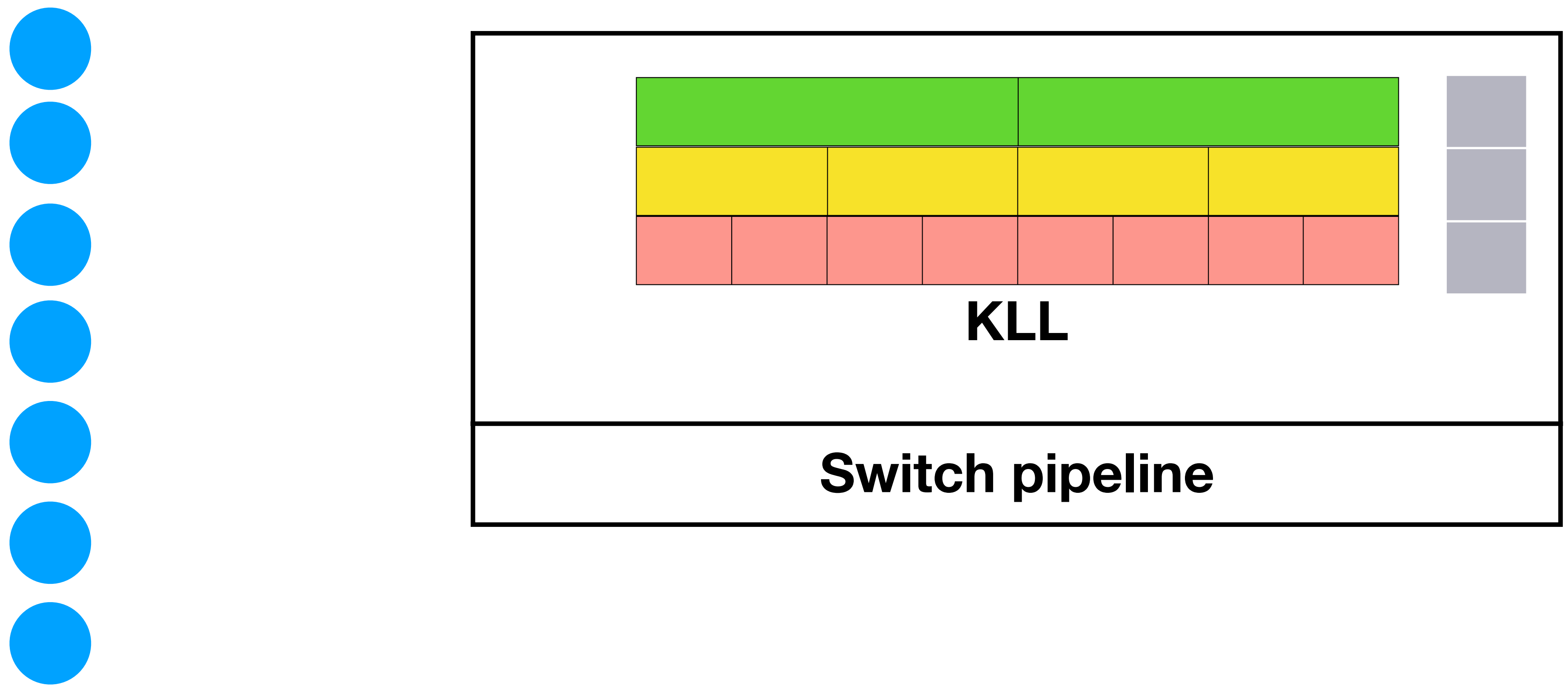
We can use **argmin()** to find the two minimum items and subsample them

Observation: Large portion of **unsampled packets** go through switch pipeline anyway

Can these unsampled packets help?

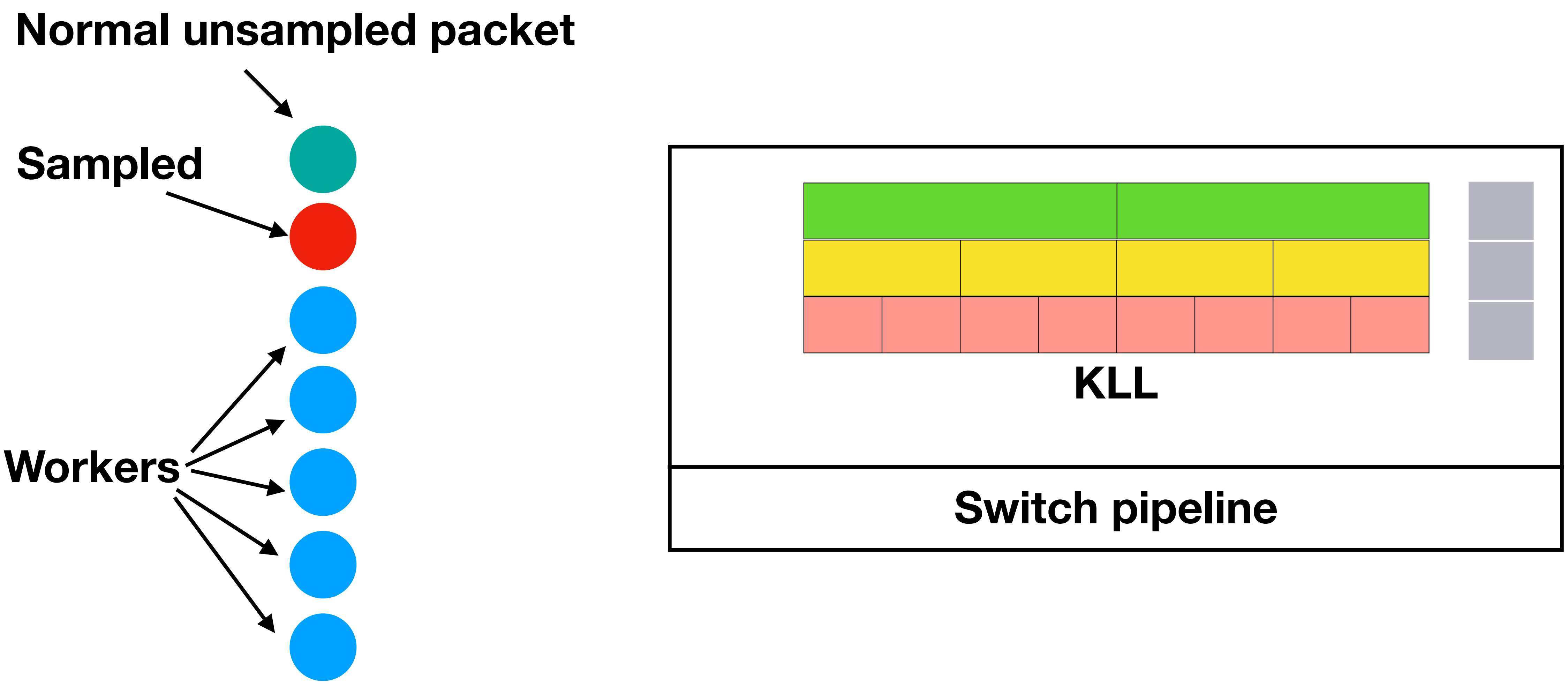
System Design

“worker packets” can help!



System Design

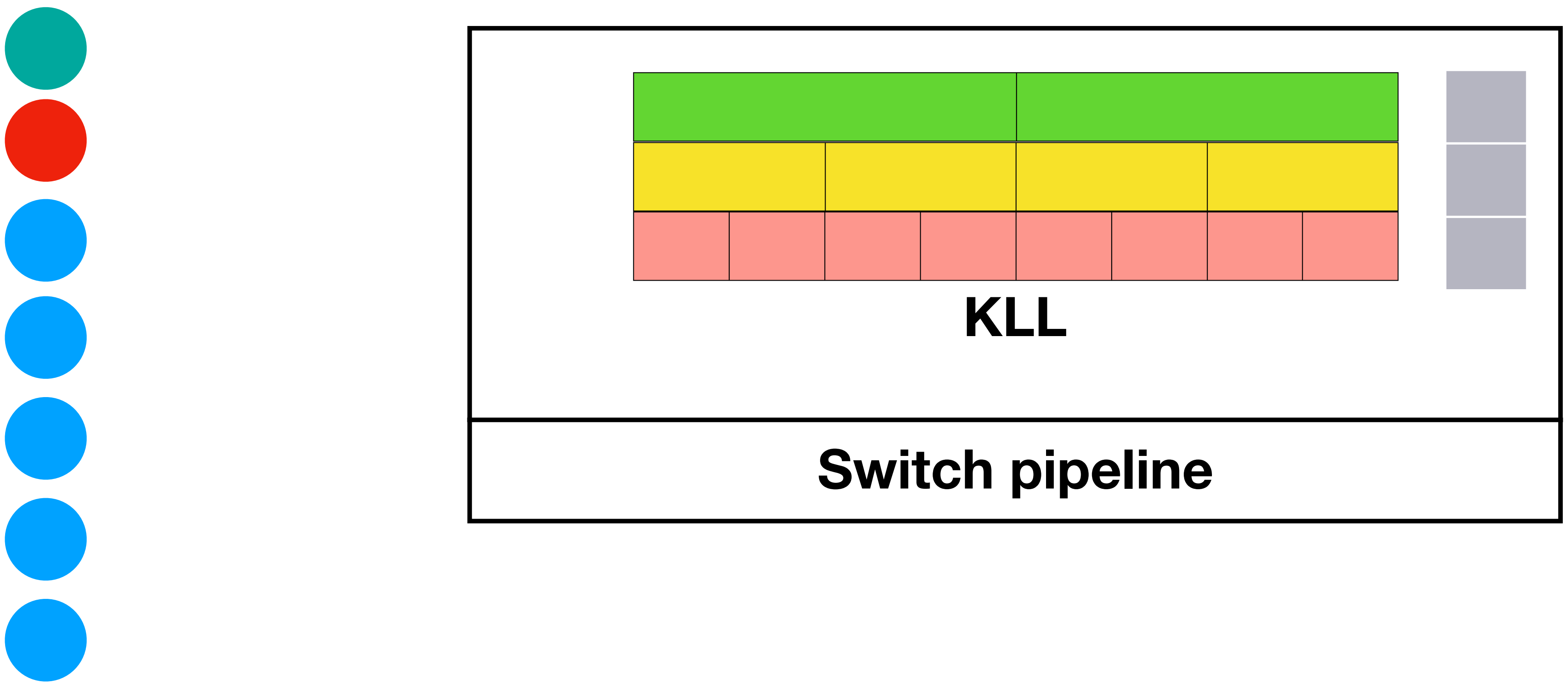
“worker packets” can help!



We want these **unsampled packets** to carry some value and help with some operations

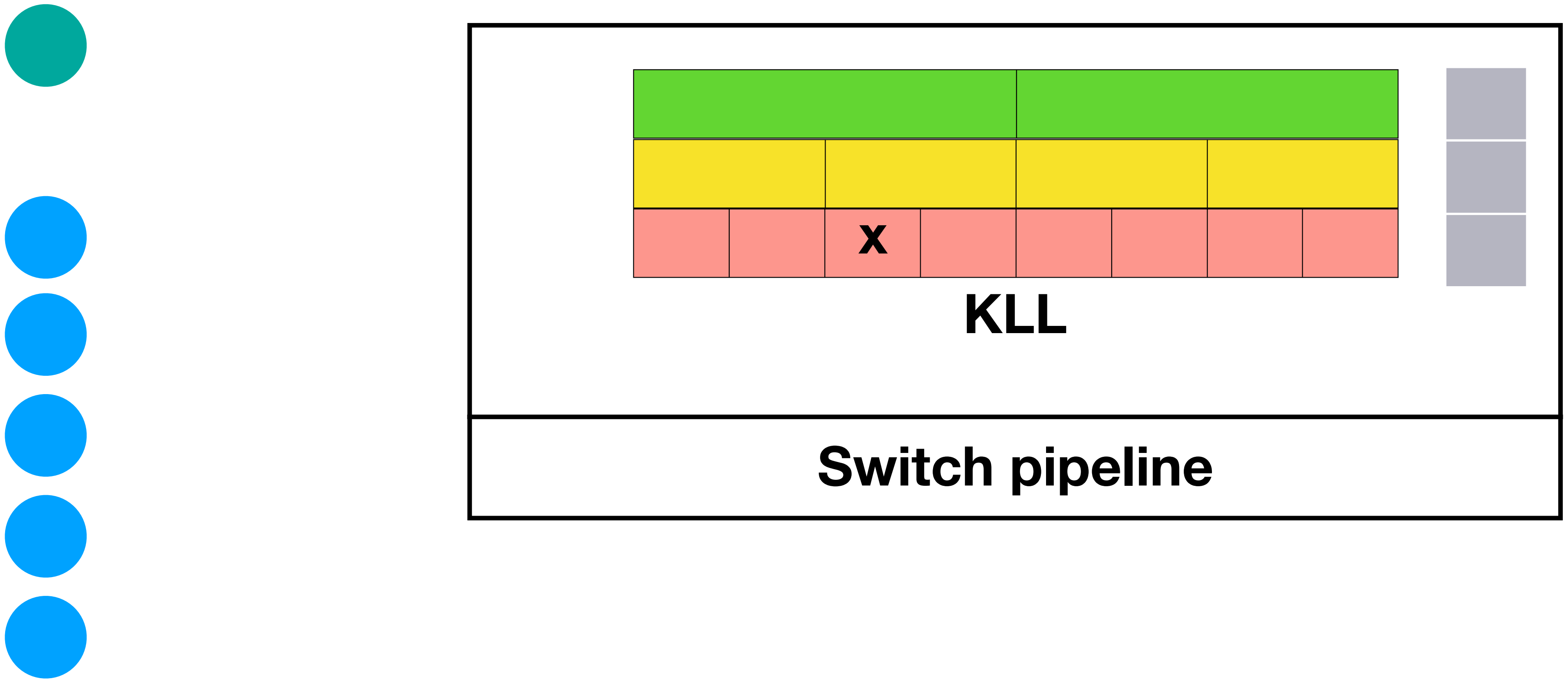
System Design

“worker packets” can help!



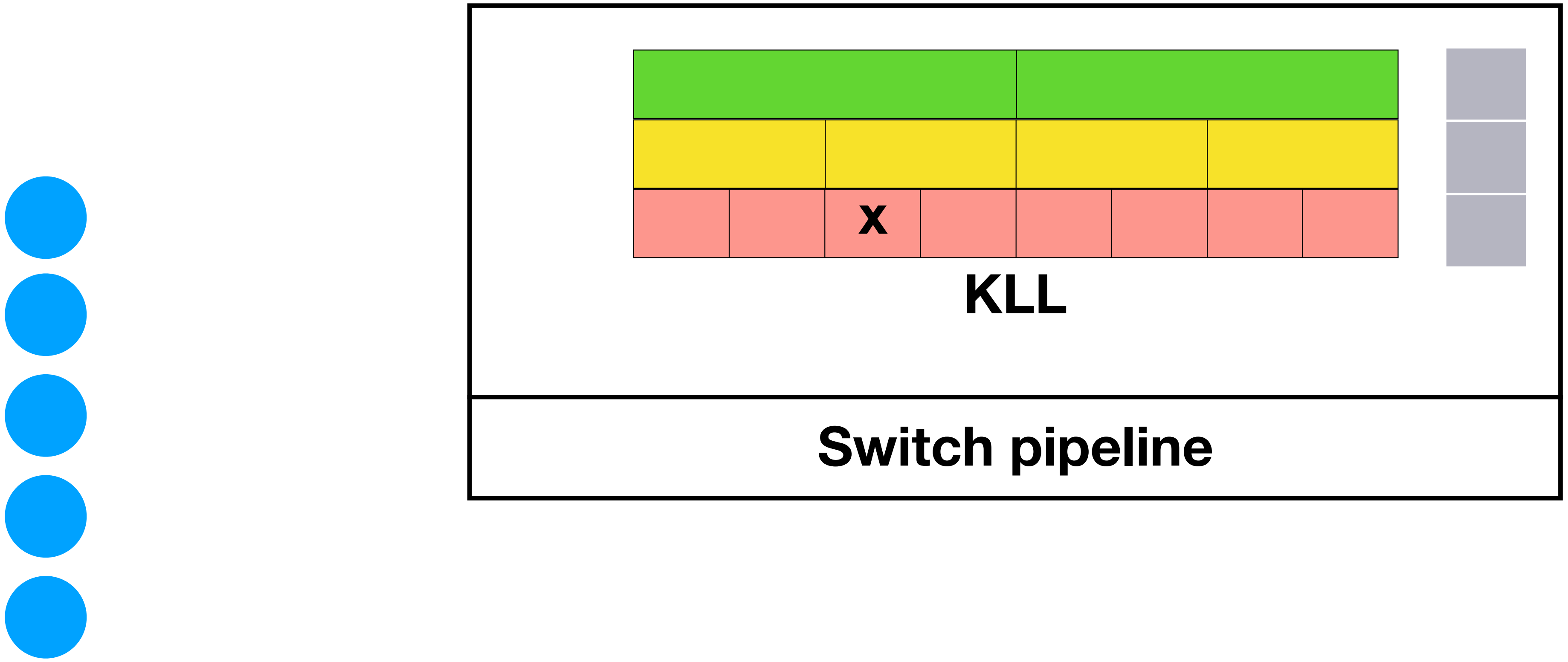
System Design

"worker packets" can help!



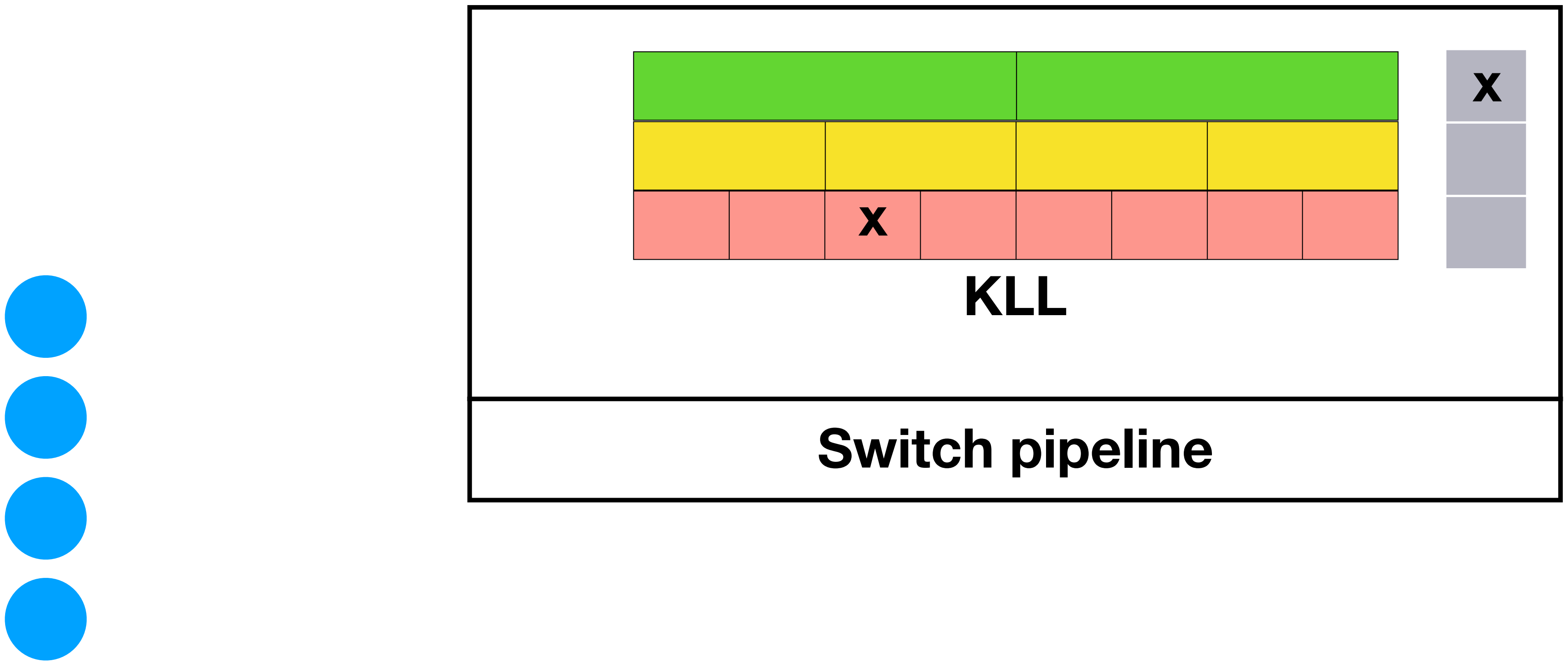
System Design

“worker packets” can help!



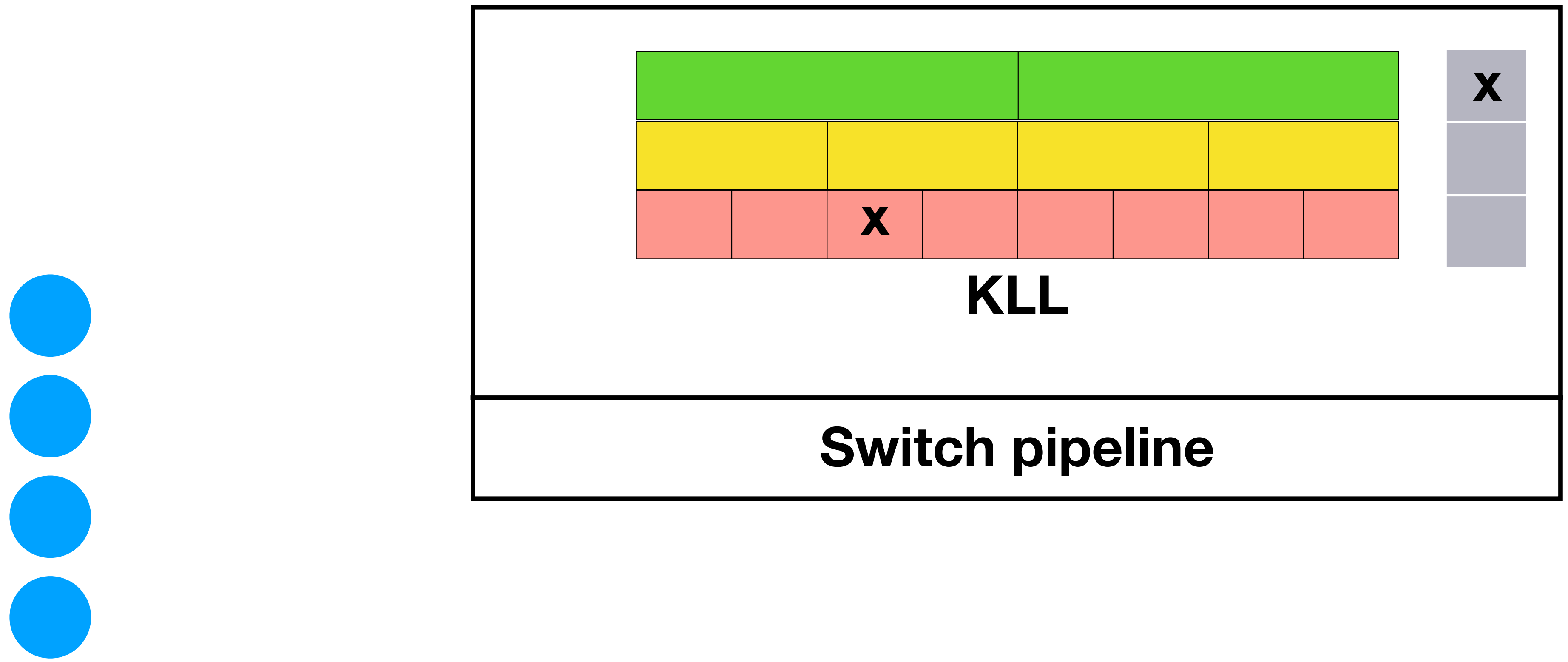
System Design

“worker packets” can help!



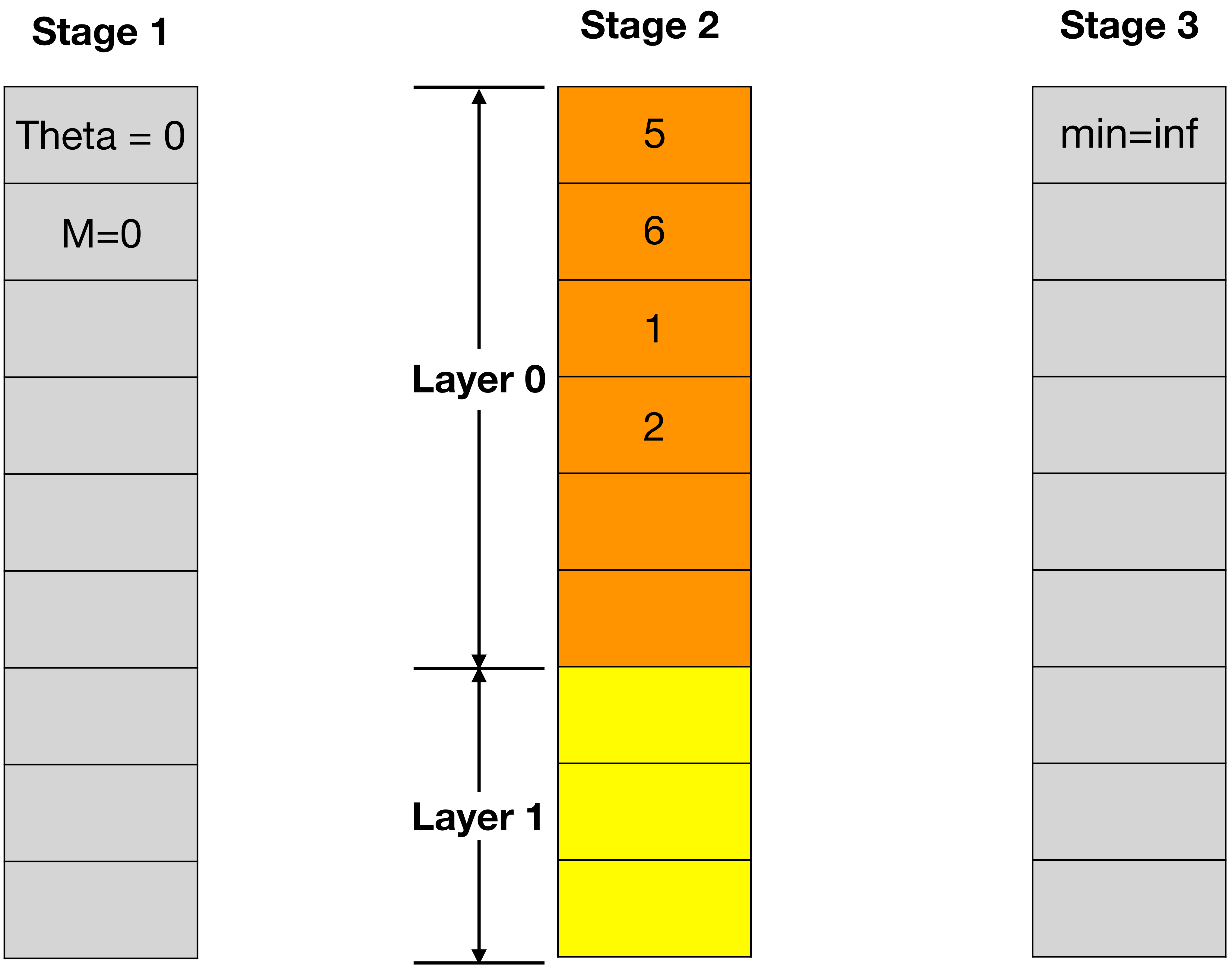
System Design

"worker packets" can help!

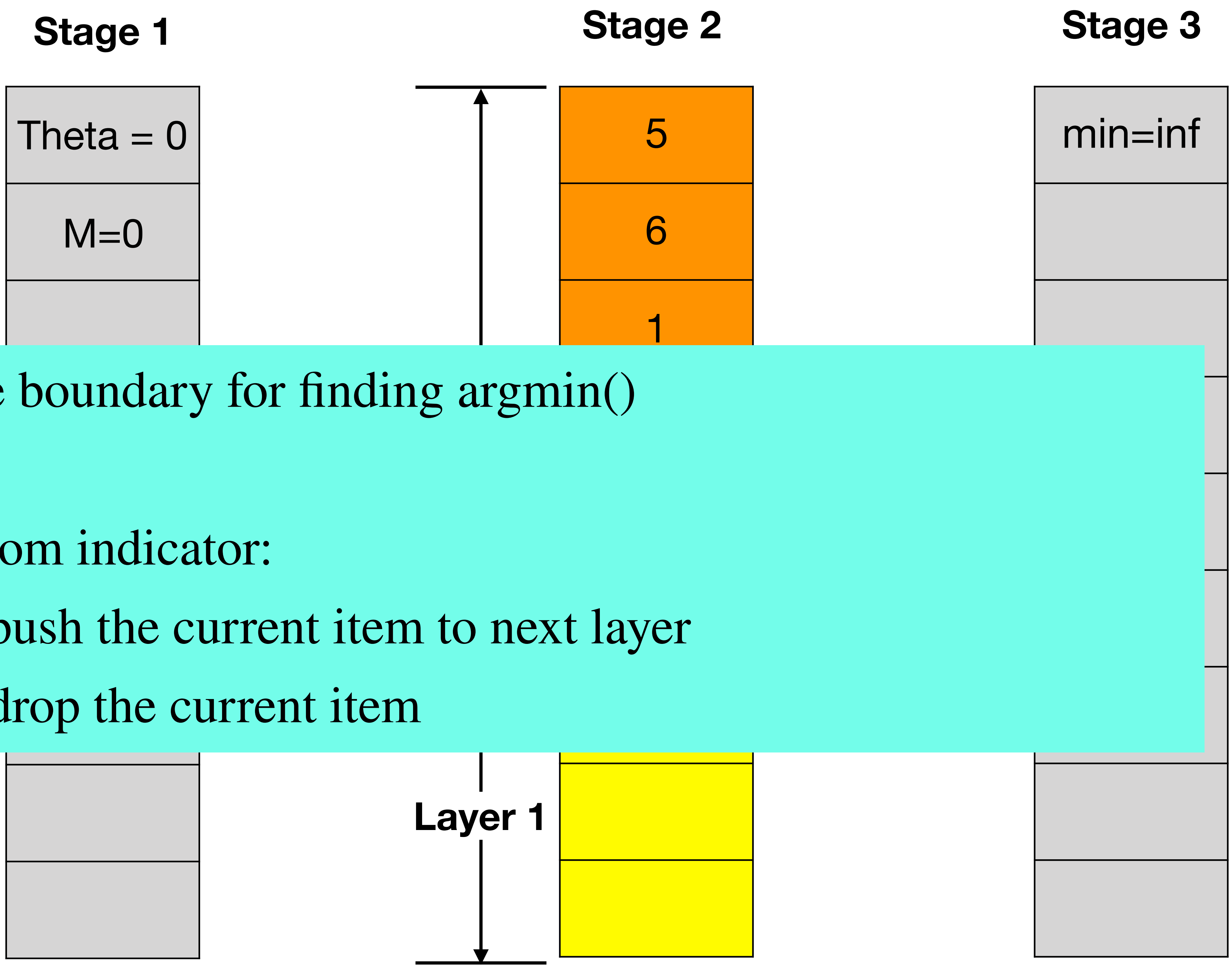


With a number of "worker packets", we can achieve many functions (e.g., argmin(), swap())

System Design



System Design

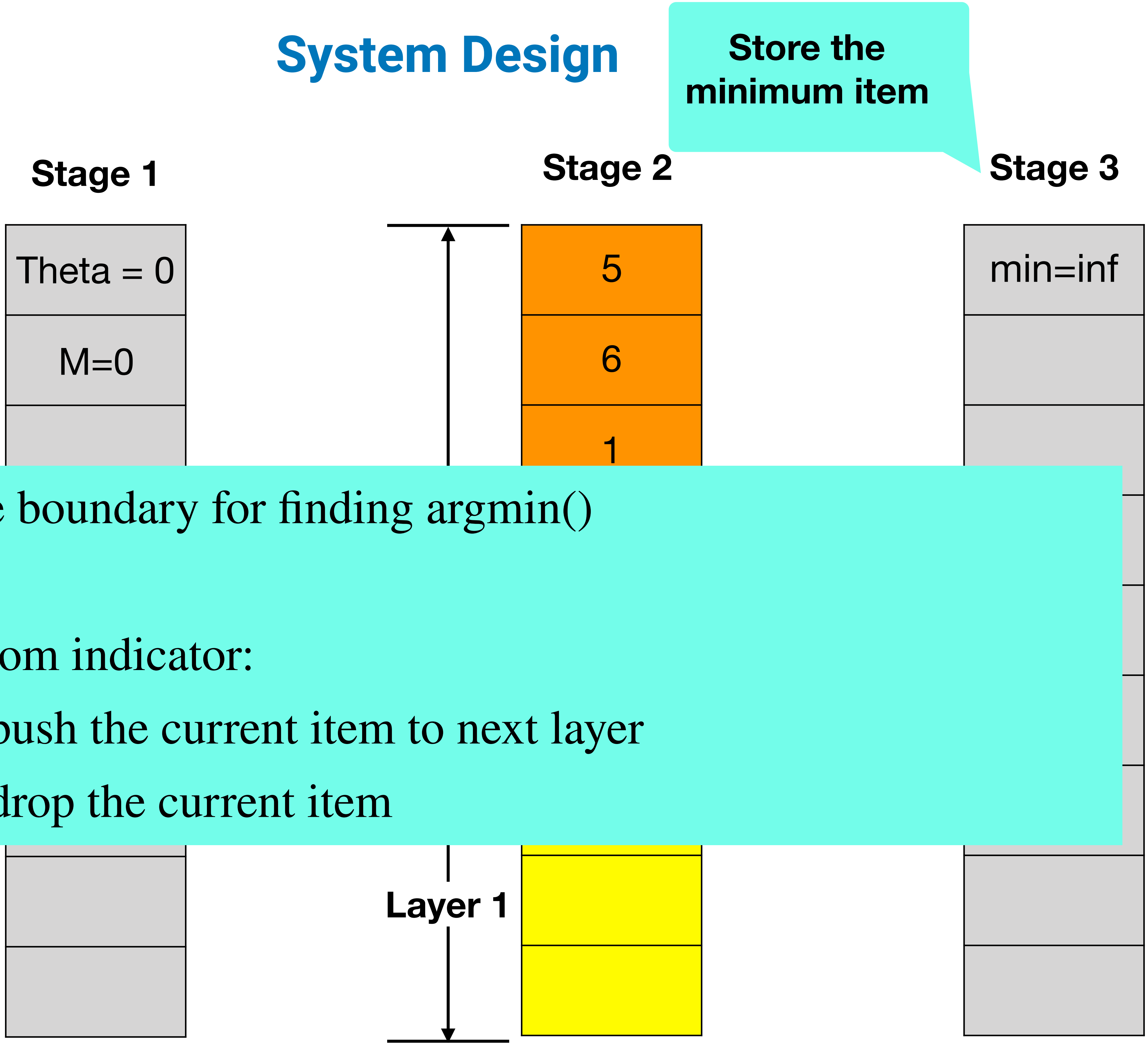


Theta is the boundary for finding argmin()

M is a random indicator:

- If M=1, push the current item to next layer
- If M=0, drop the current item

System Design

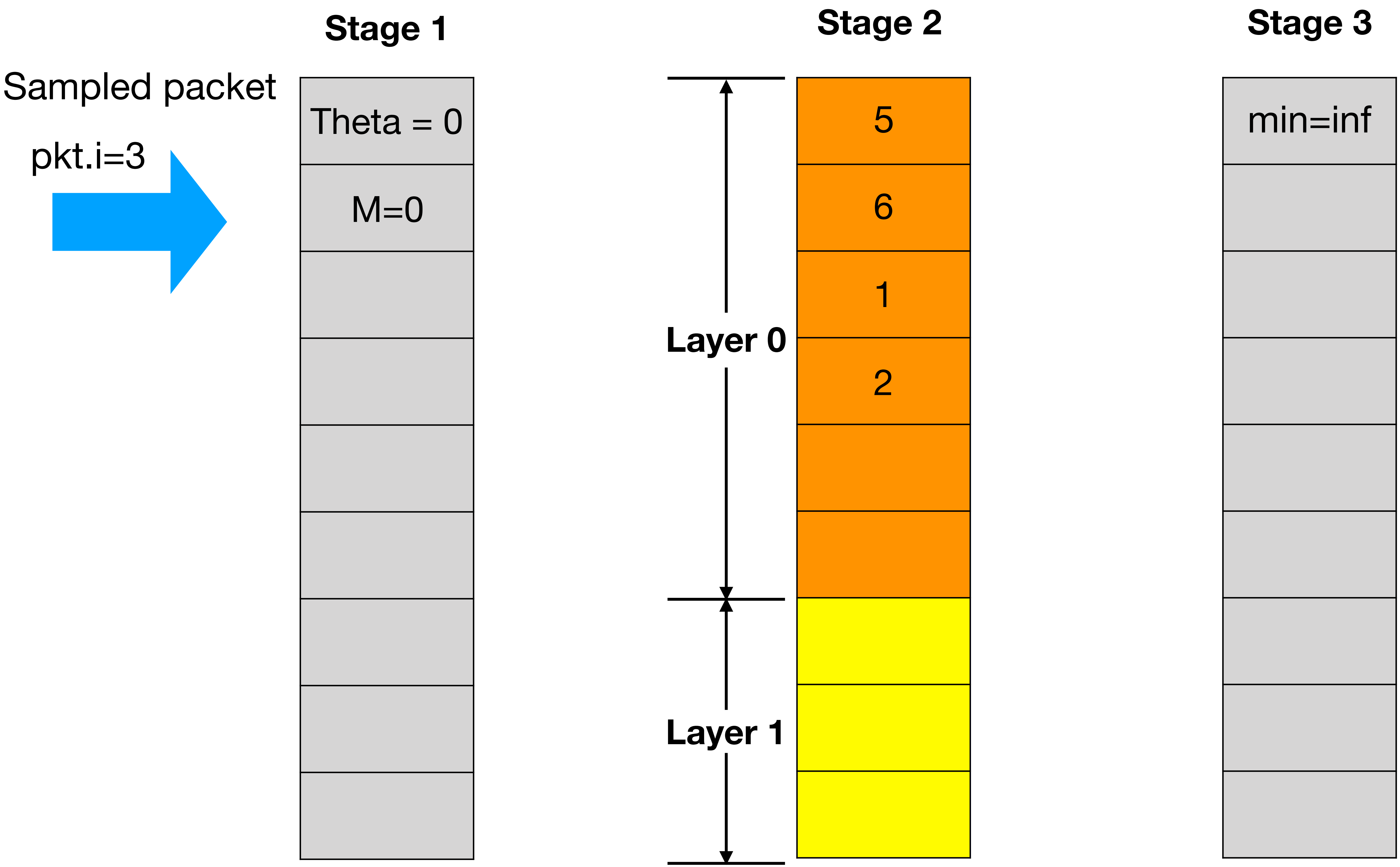


Theta is the boundary for finding argmin()

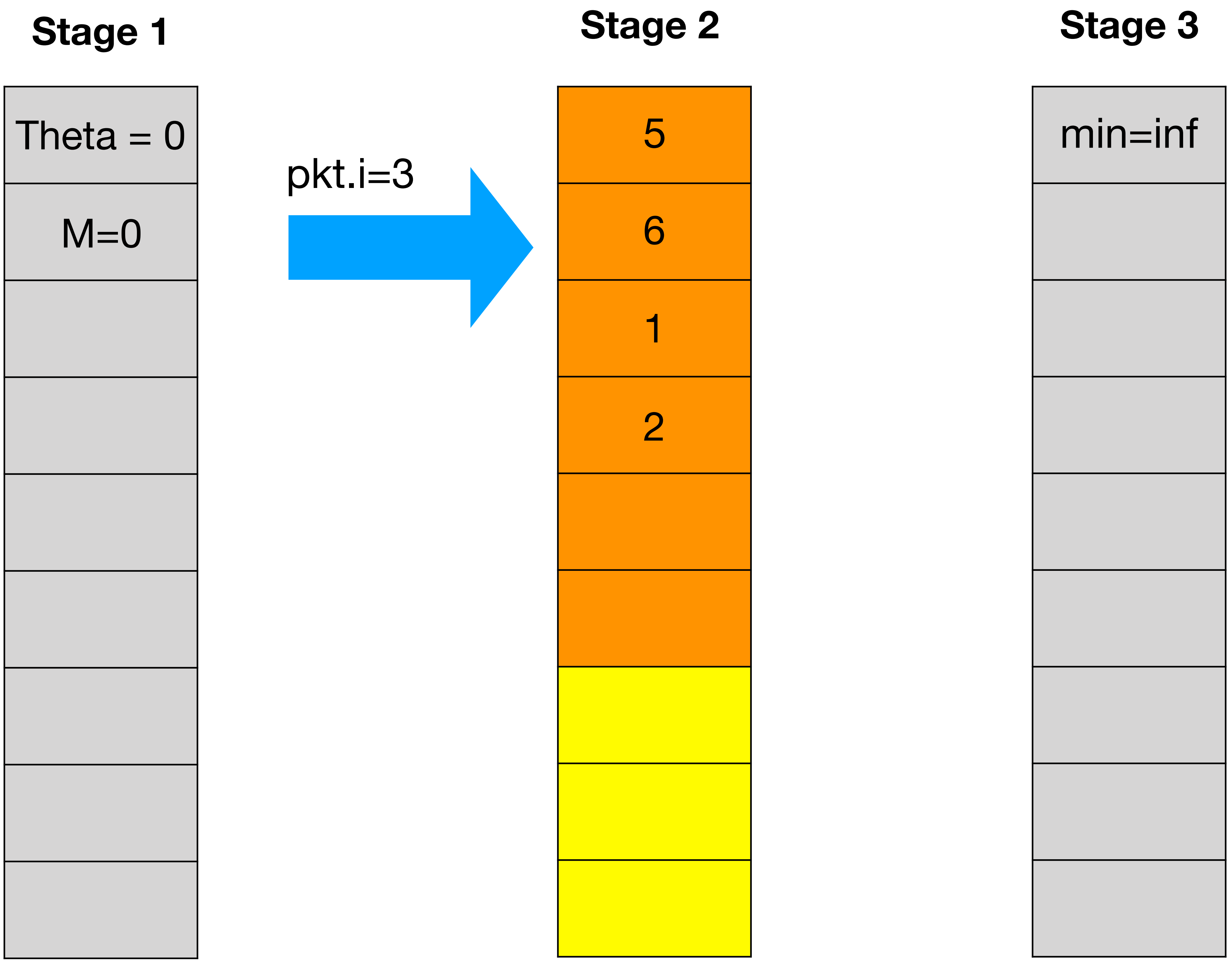
M is a random indicator:

- If M=1, push the current item to next layer
- If M=0, drop the current item

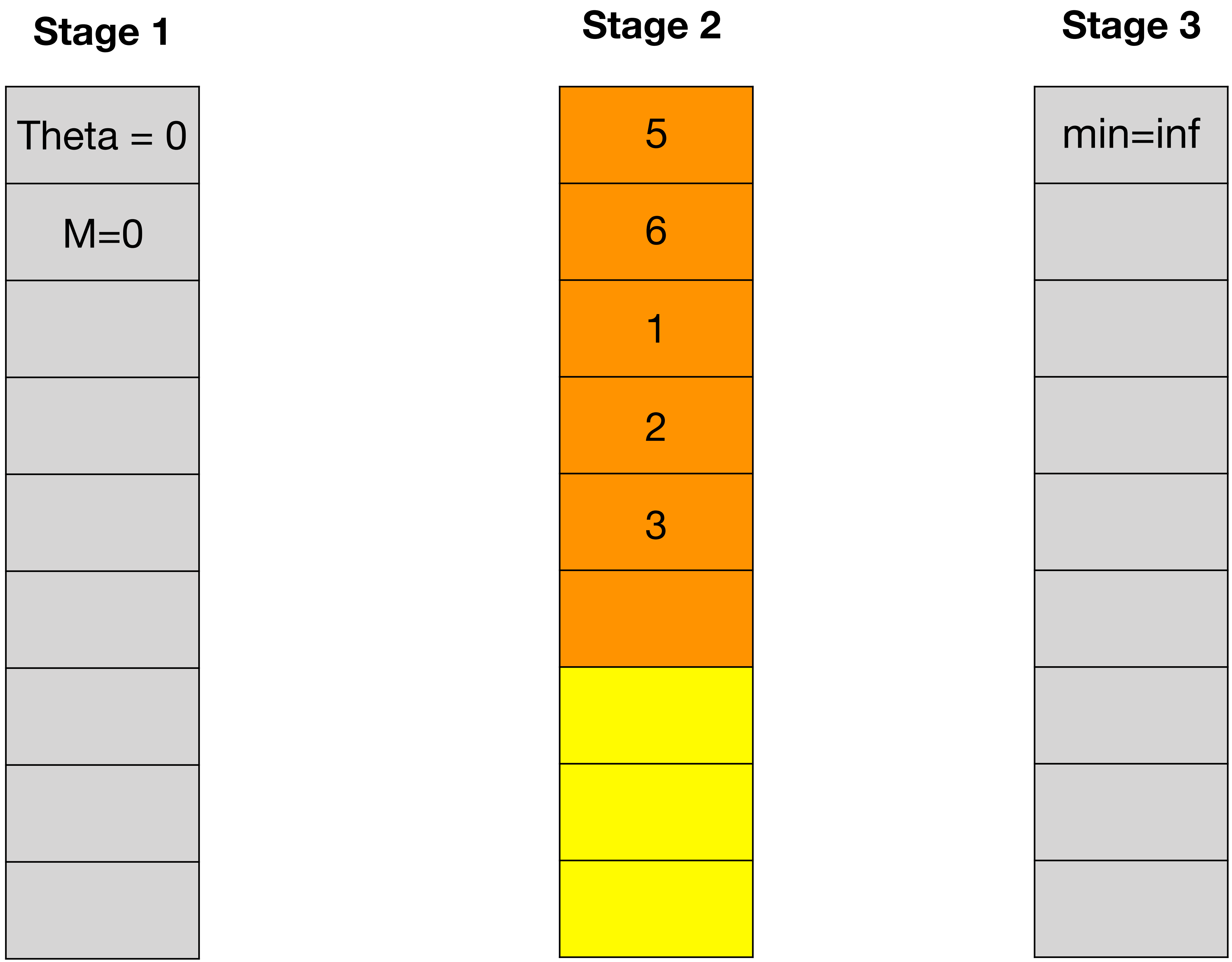
System Design



System Design

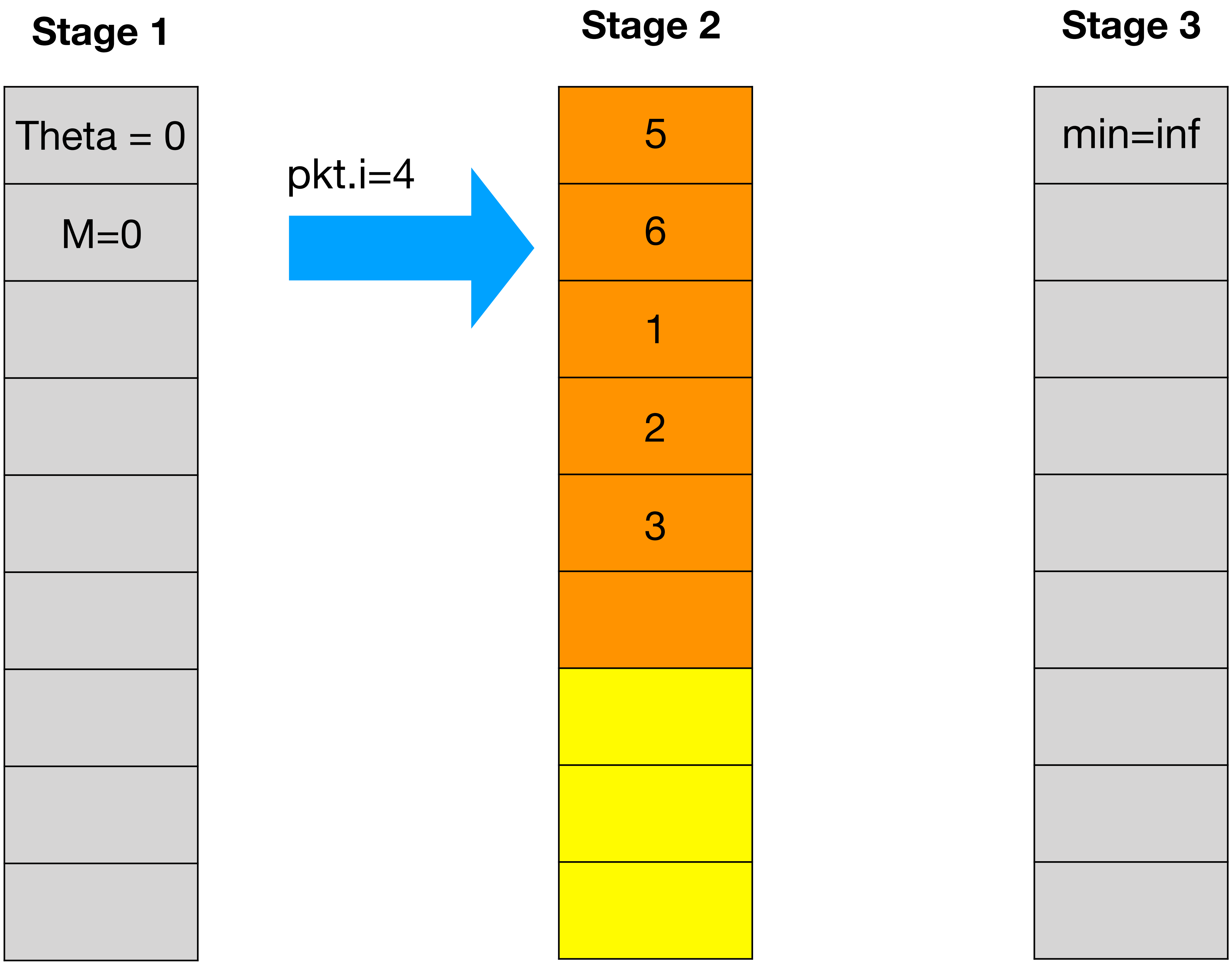


System Design

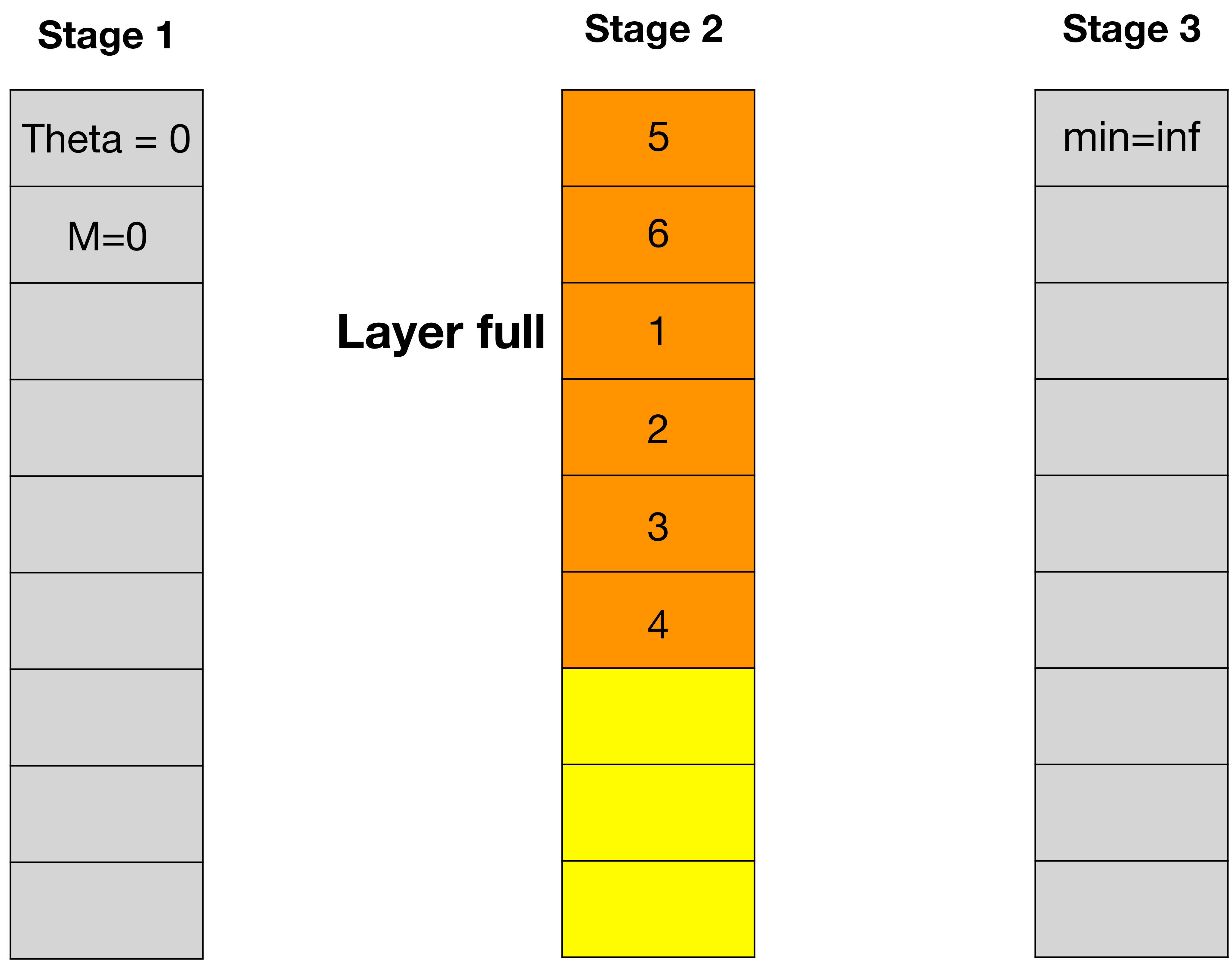


System Design

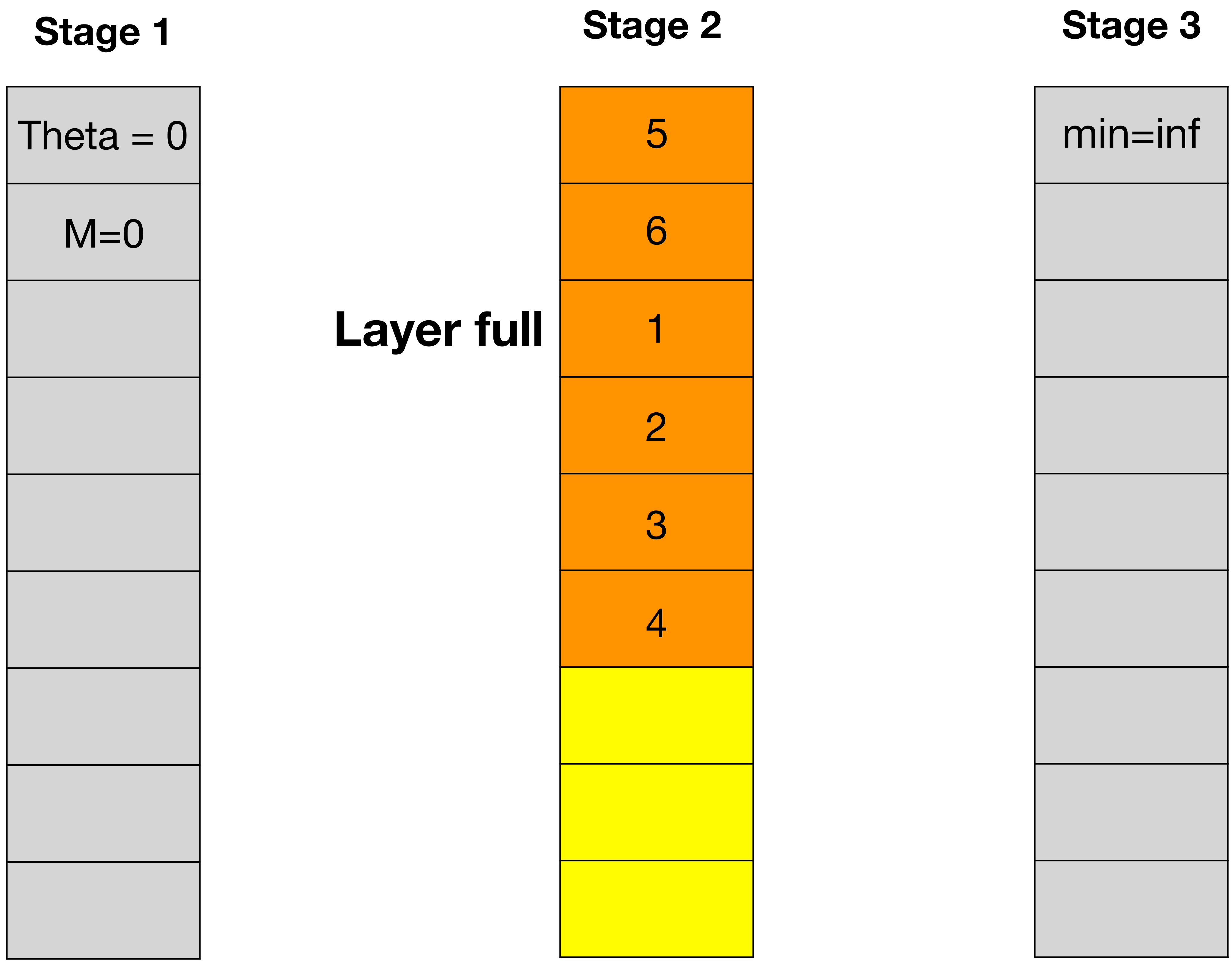
System Design



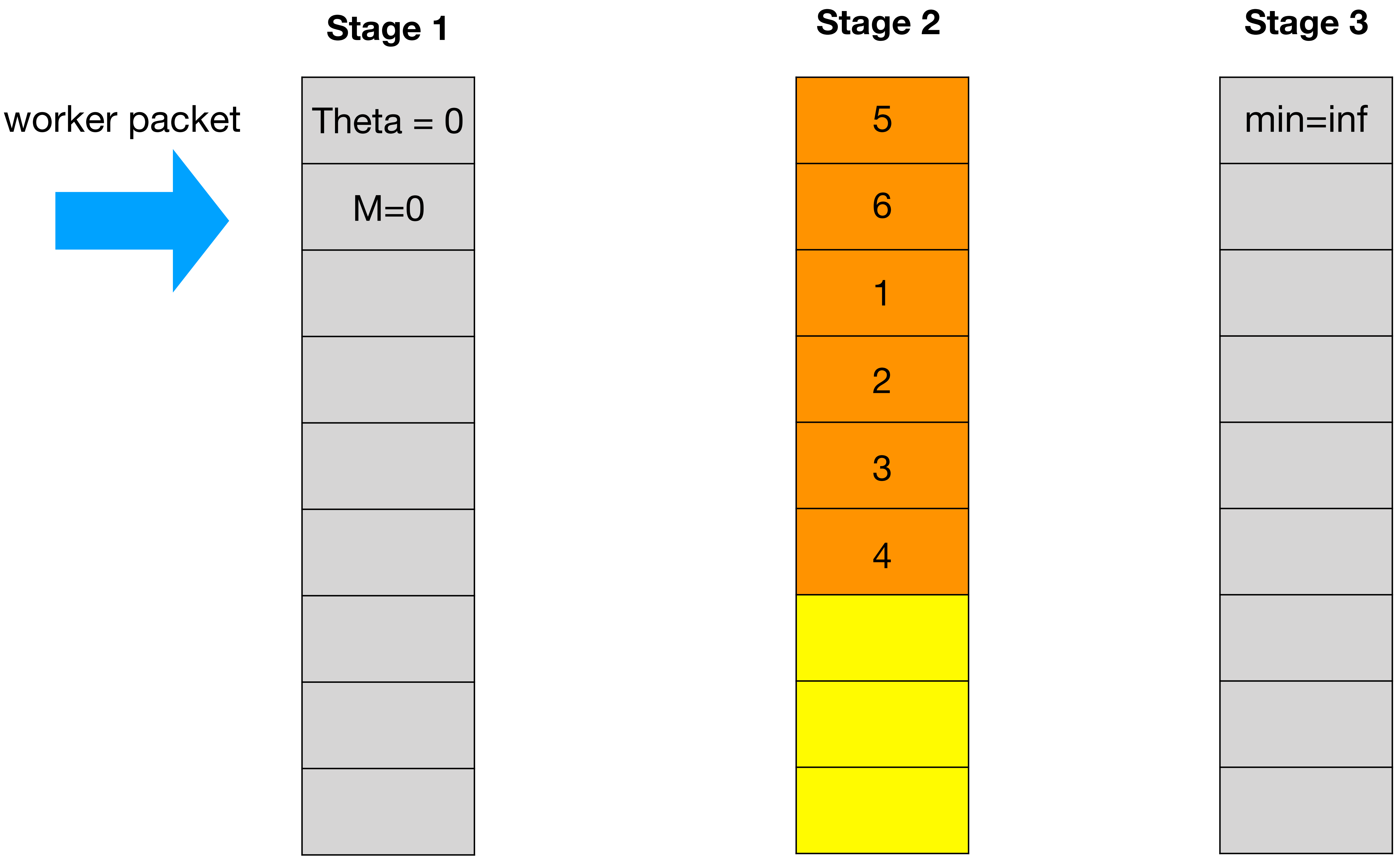
System Design



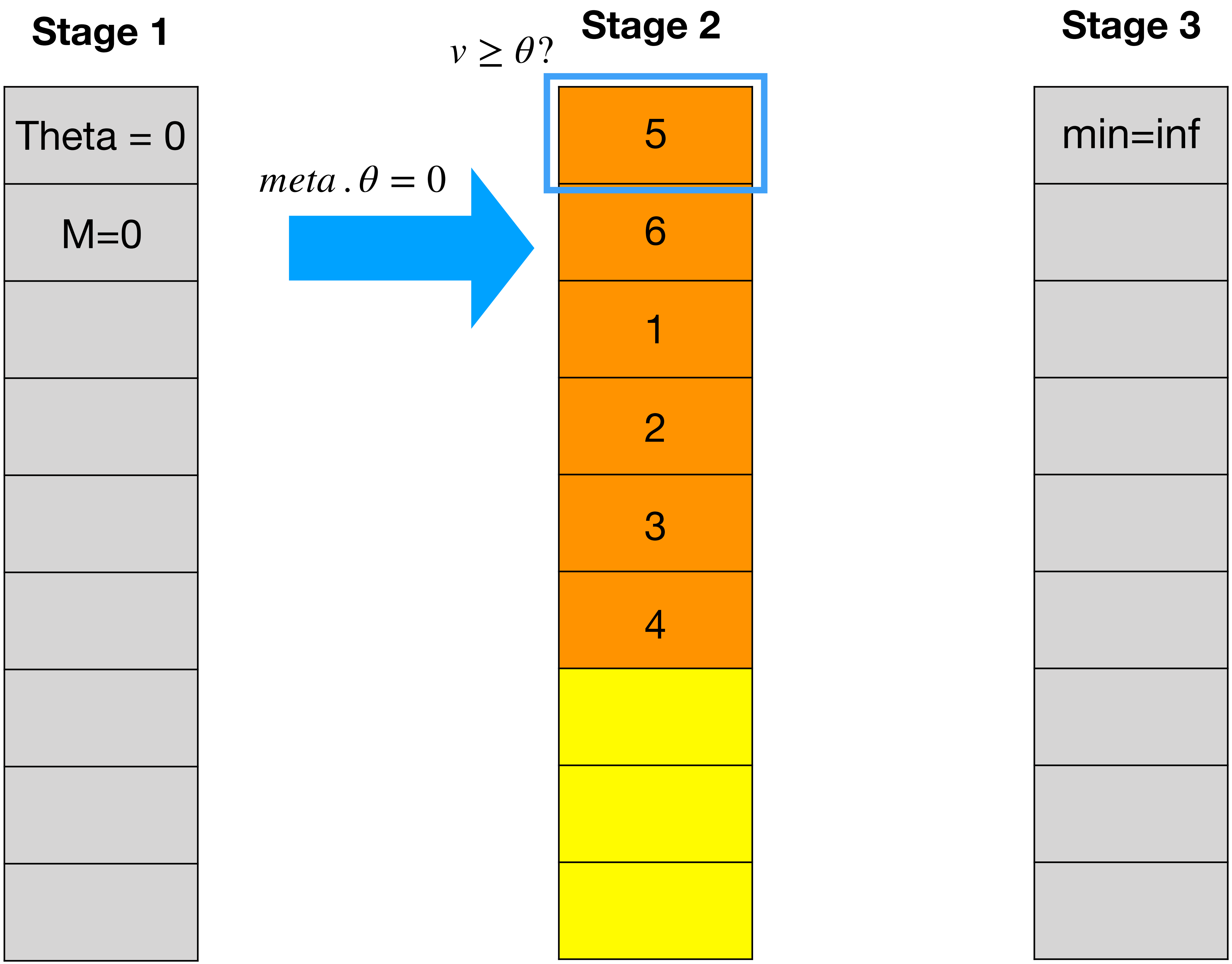
System Design



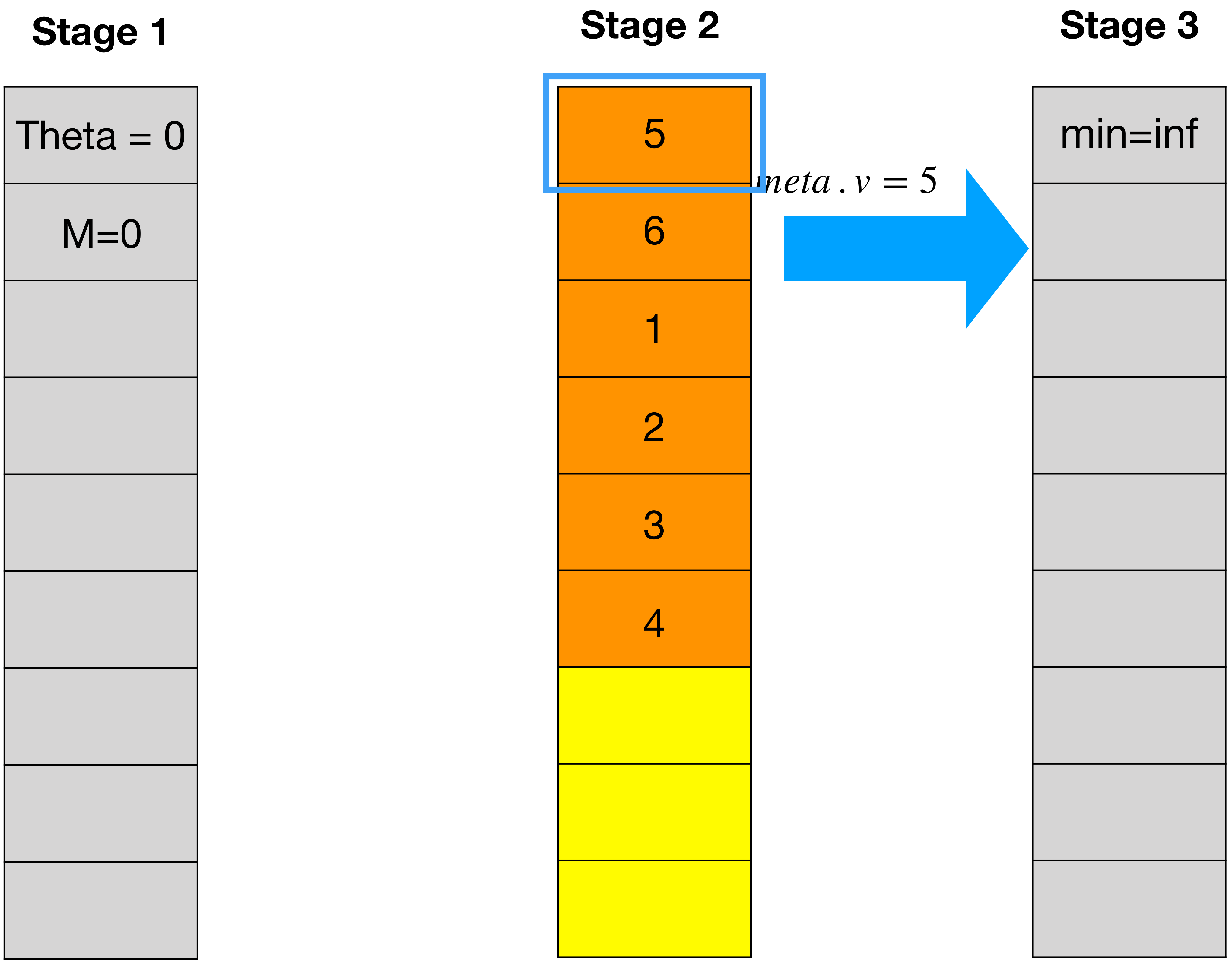
System Design



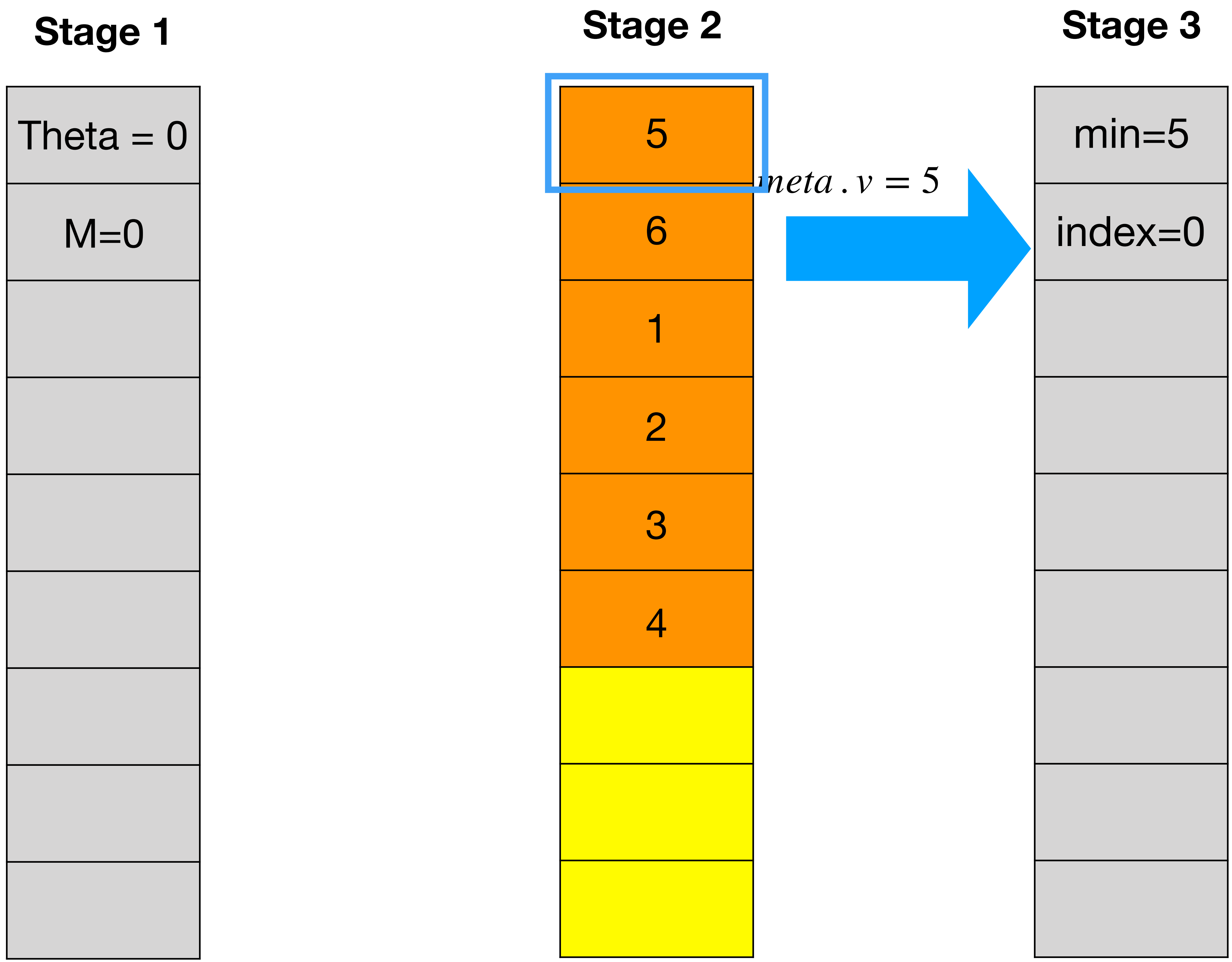
System Design



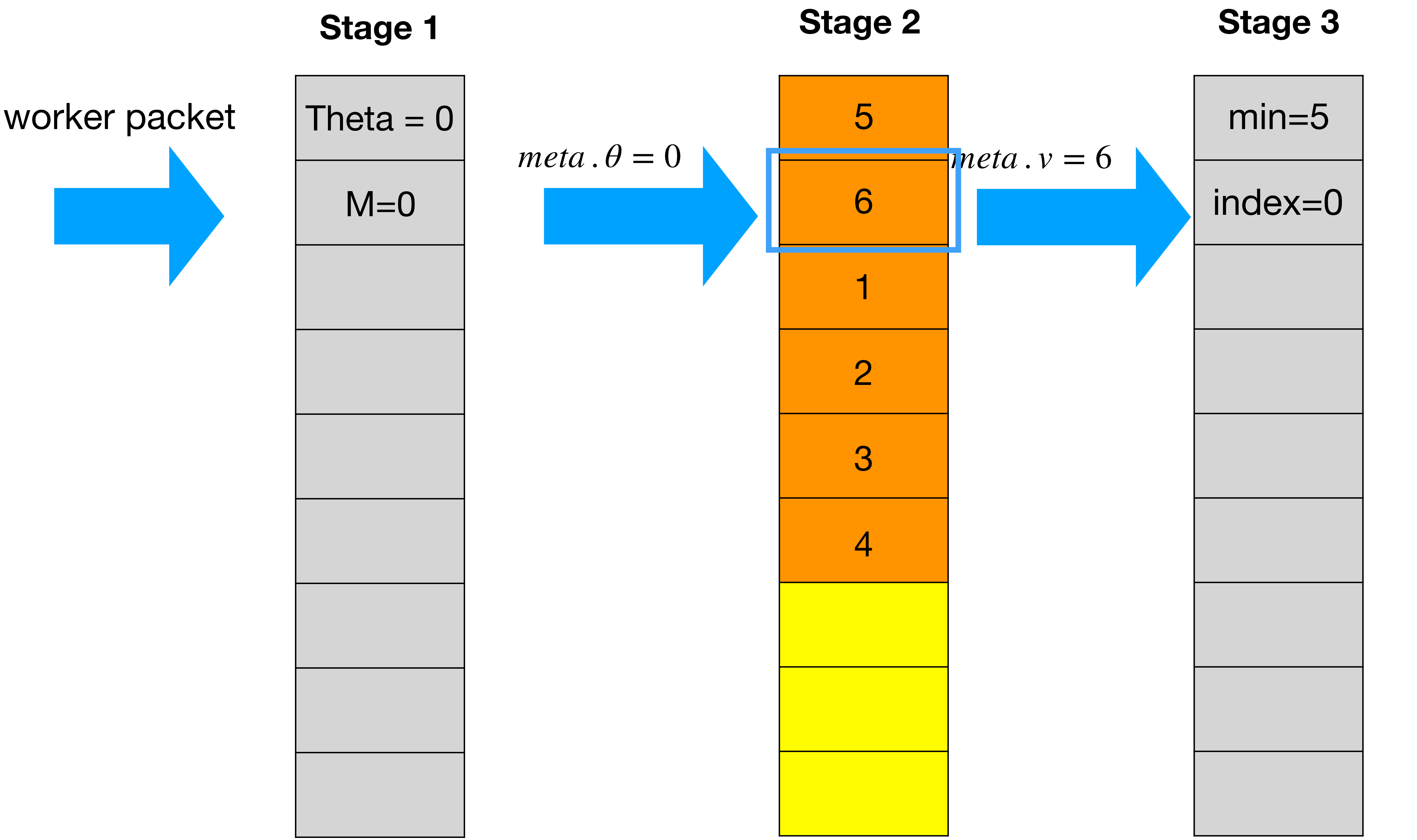
System Design



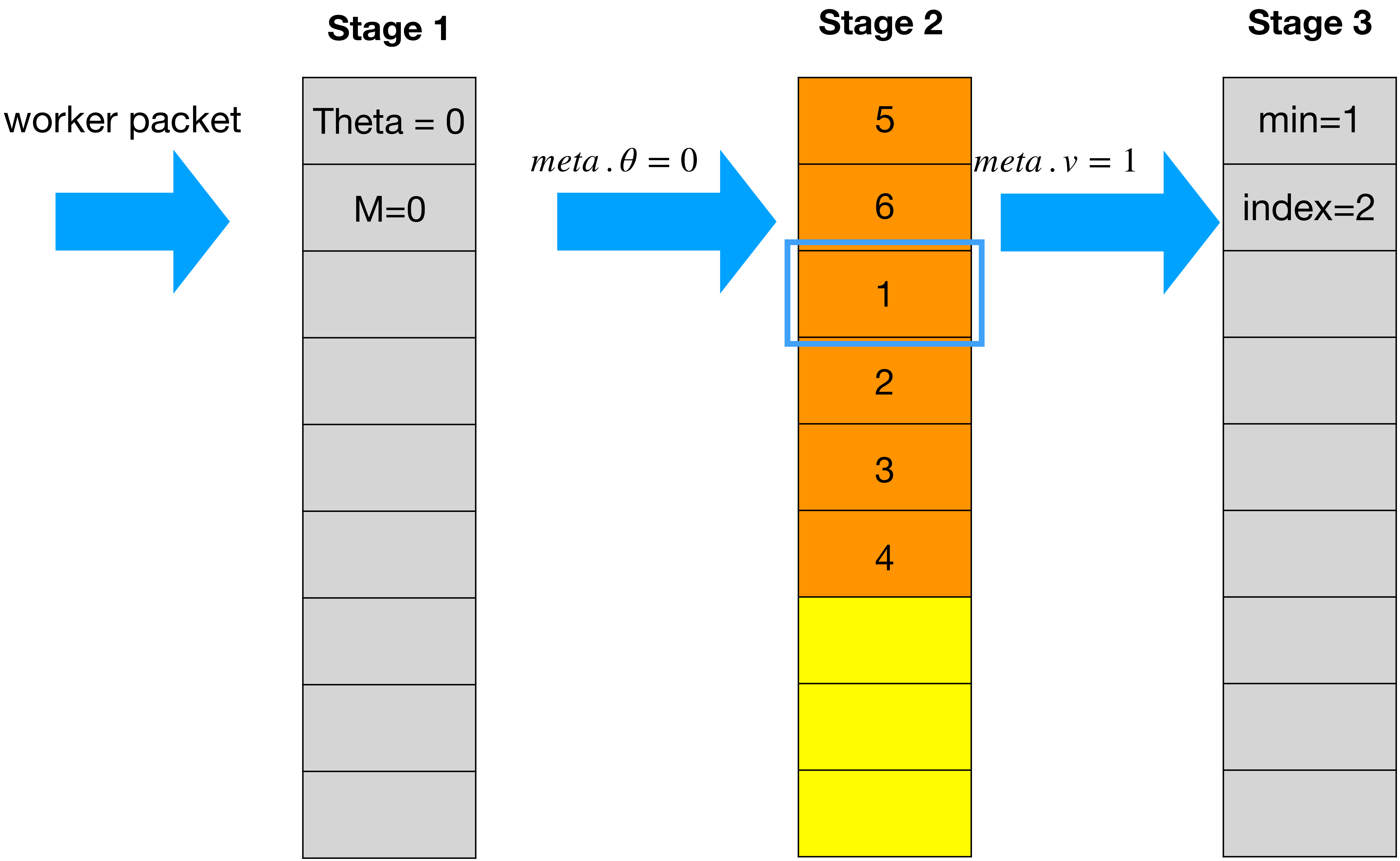
System Design



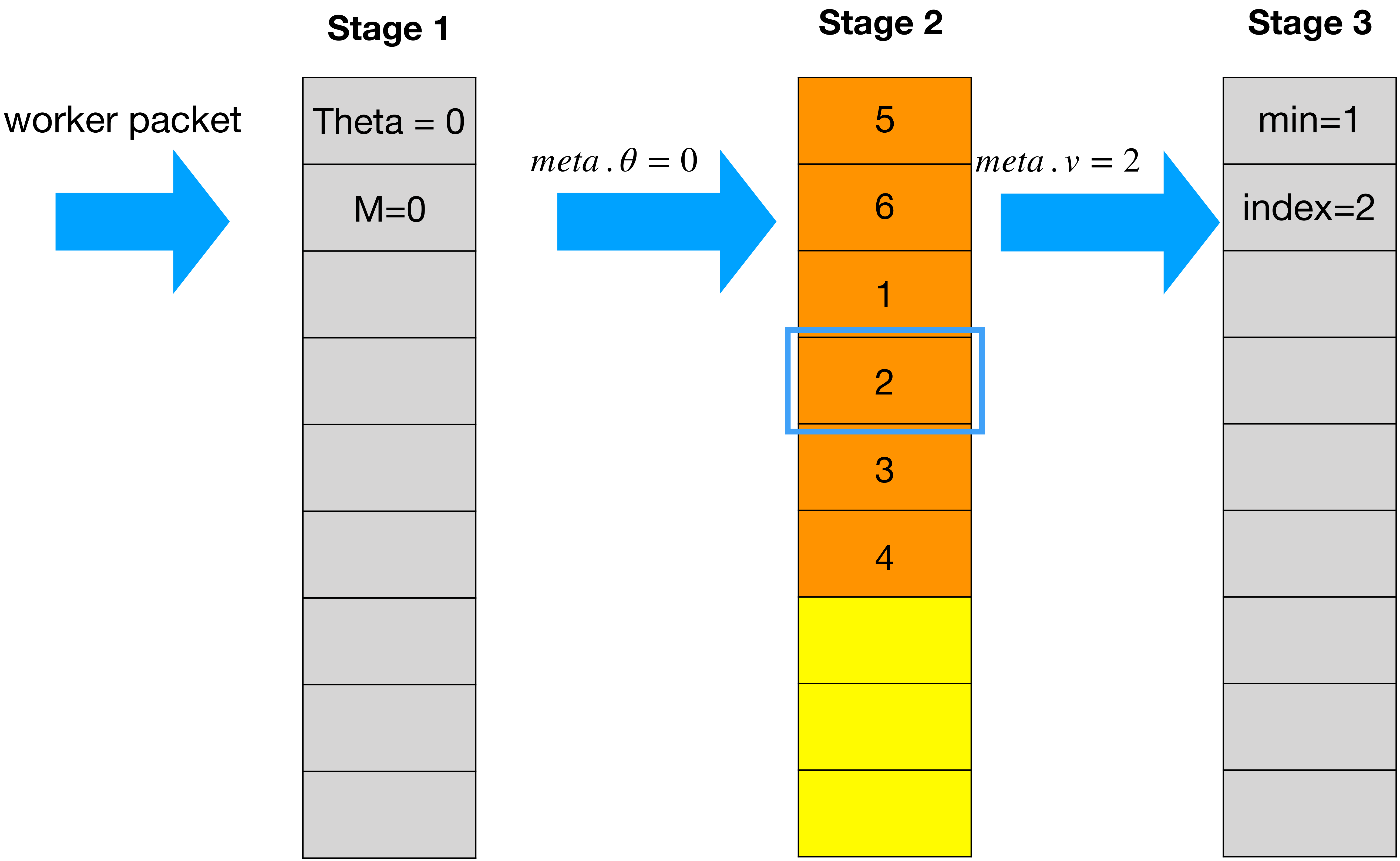
System Design



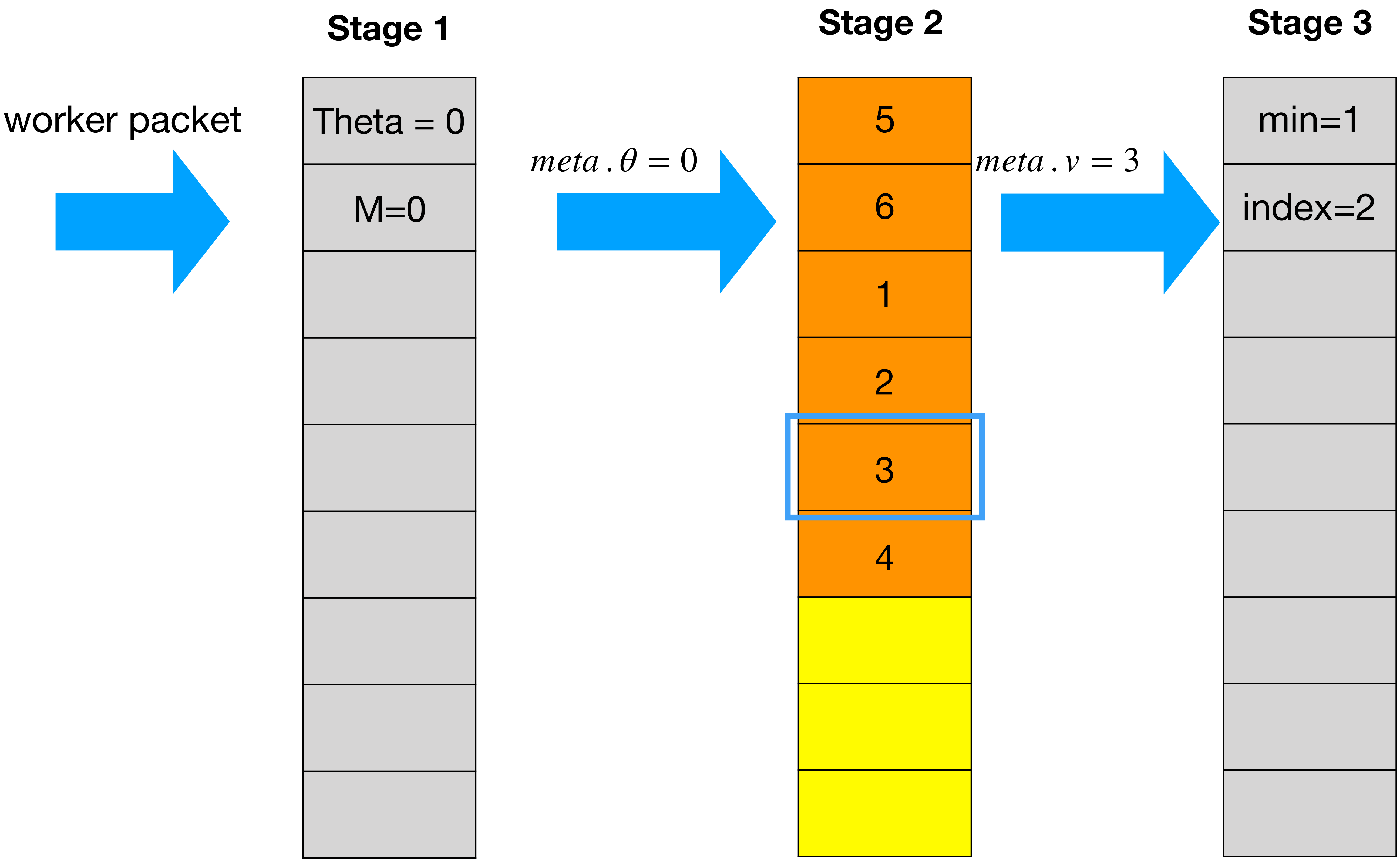
System Design



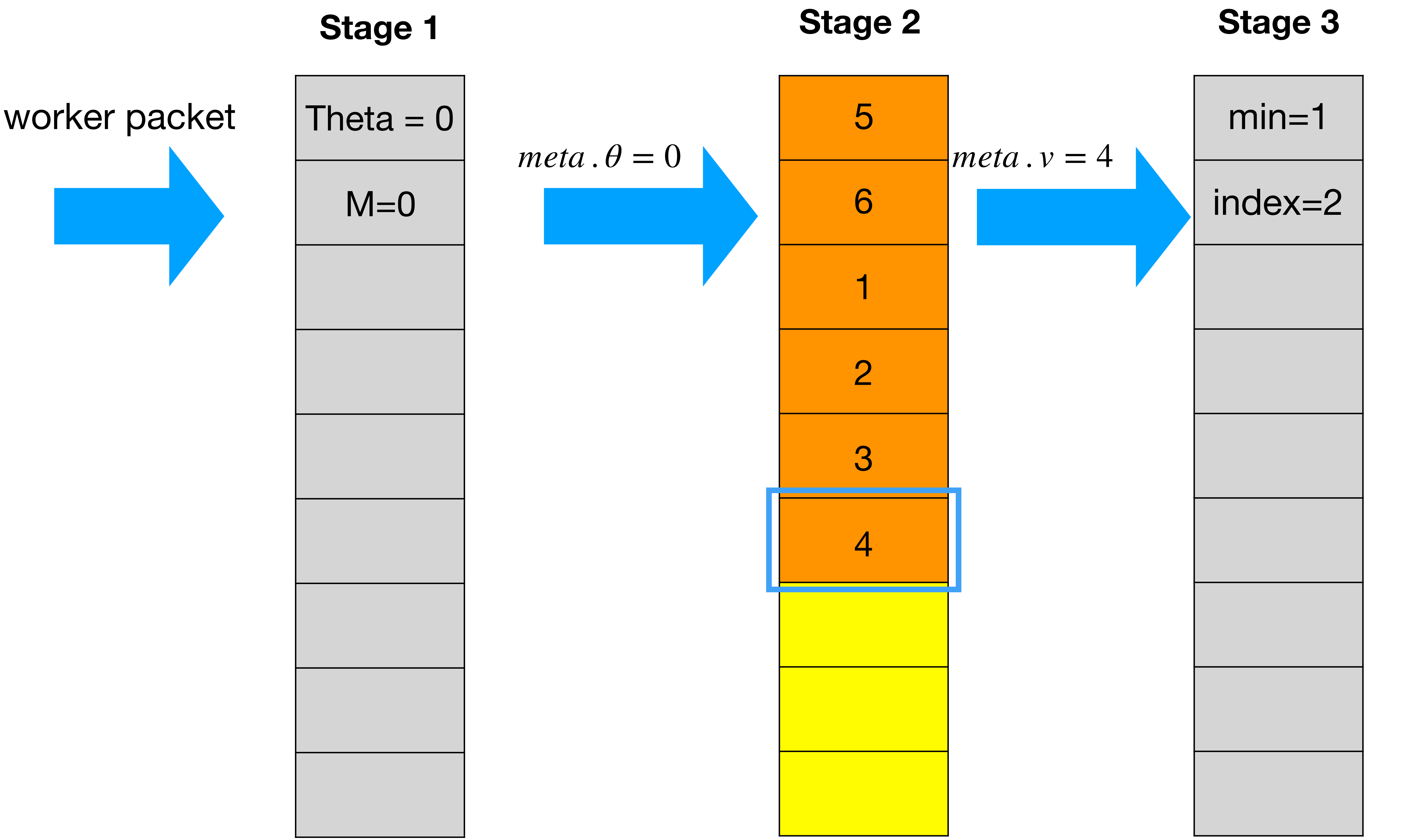
System Design



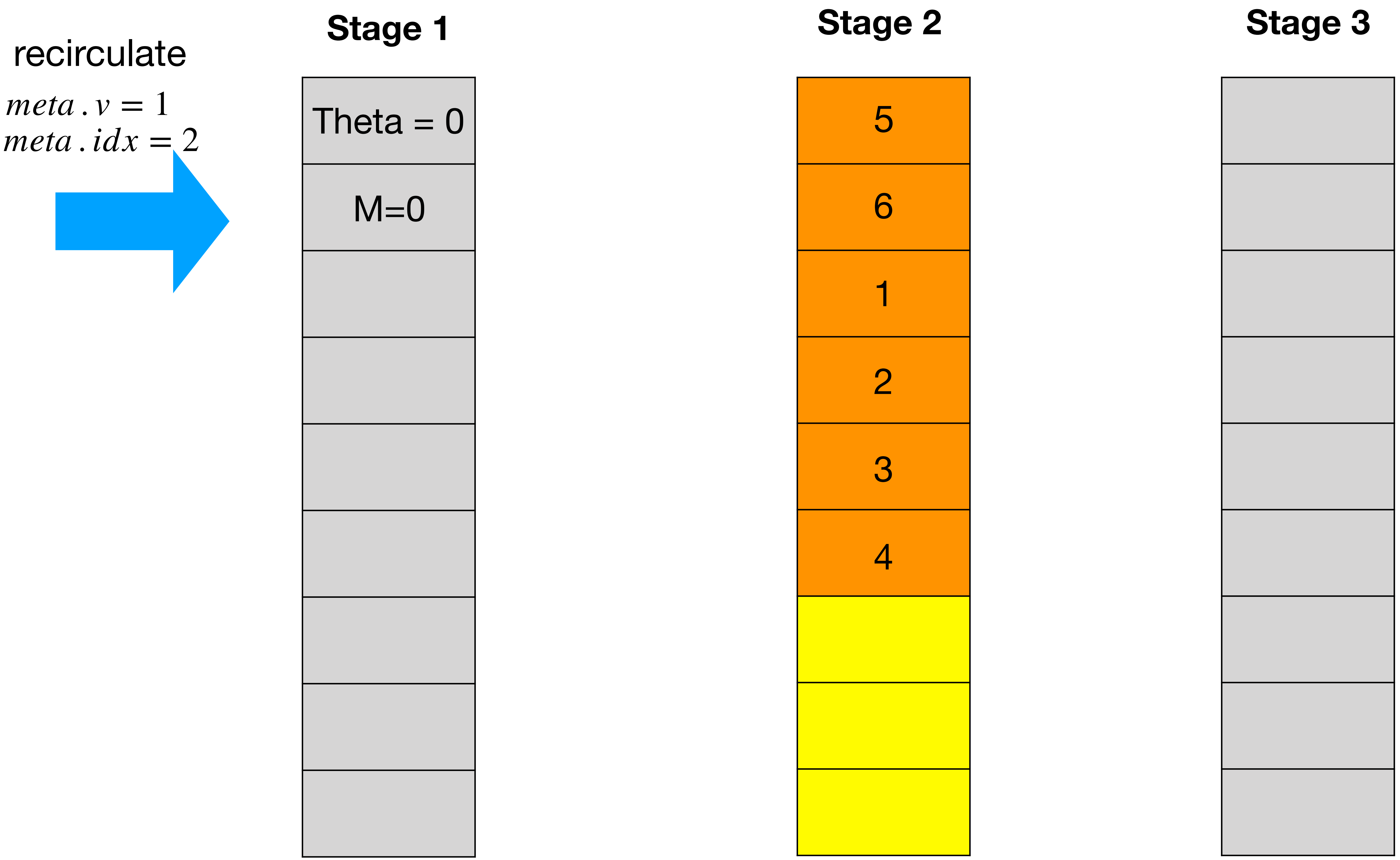
System Design



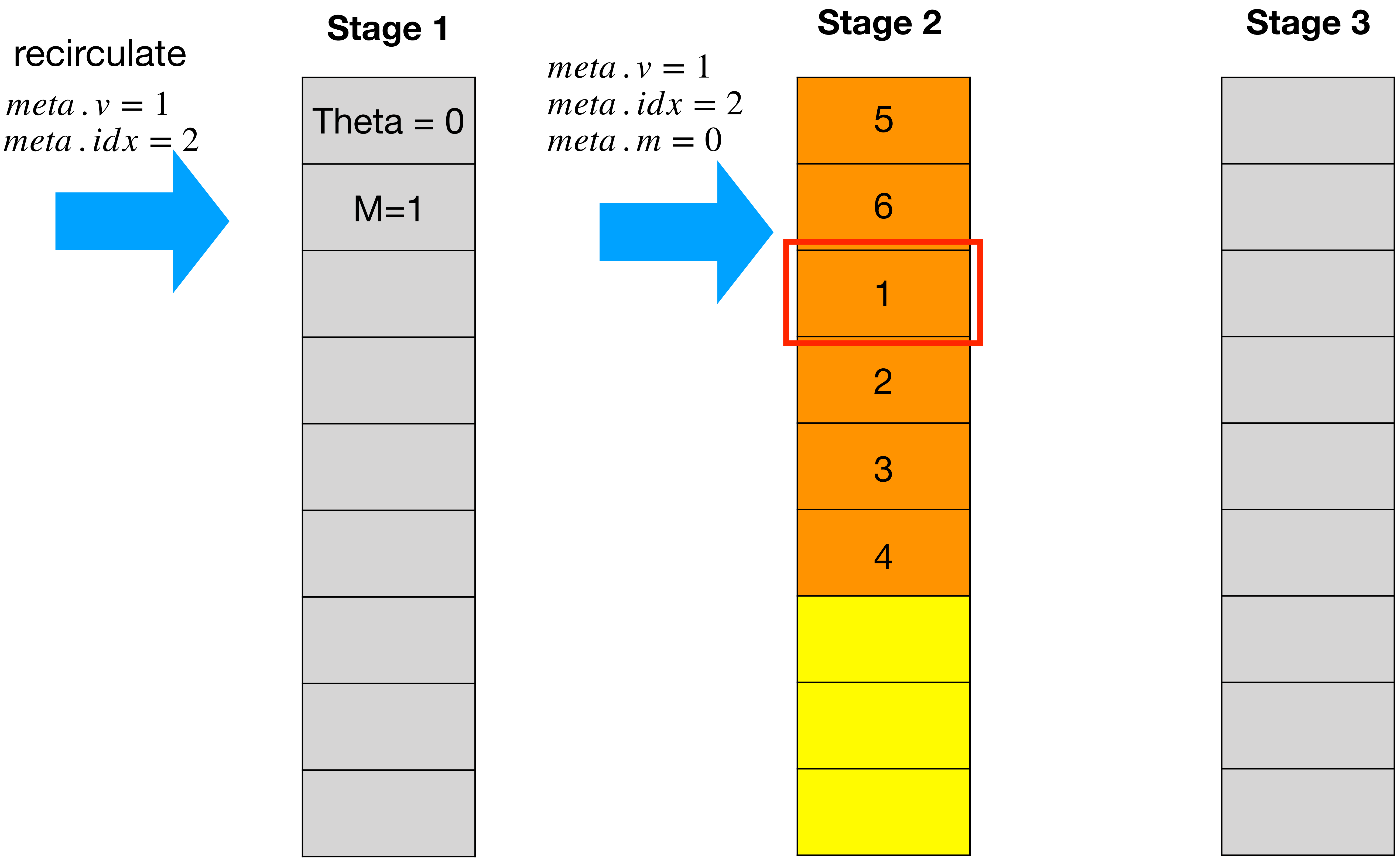
System Design



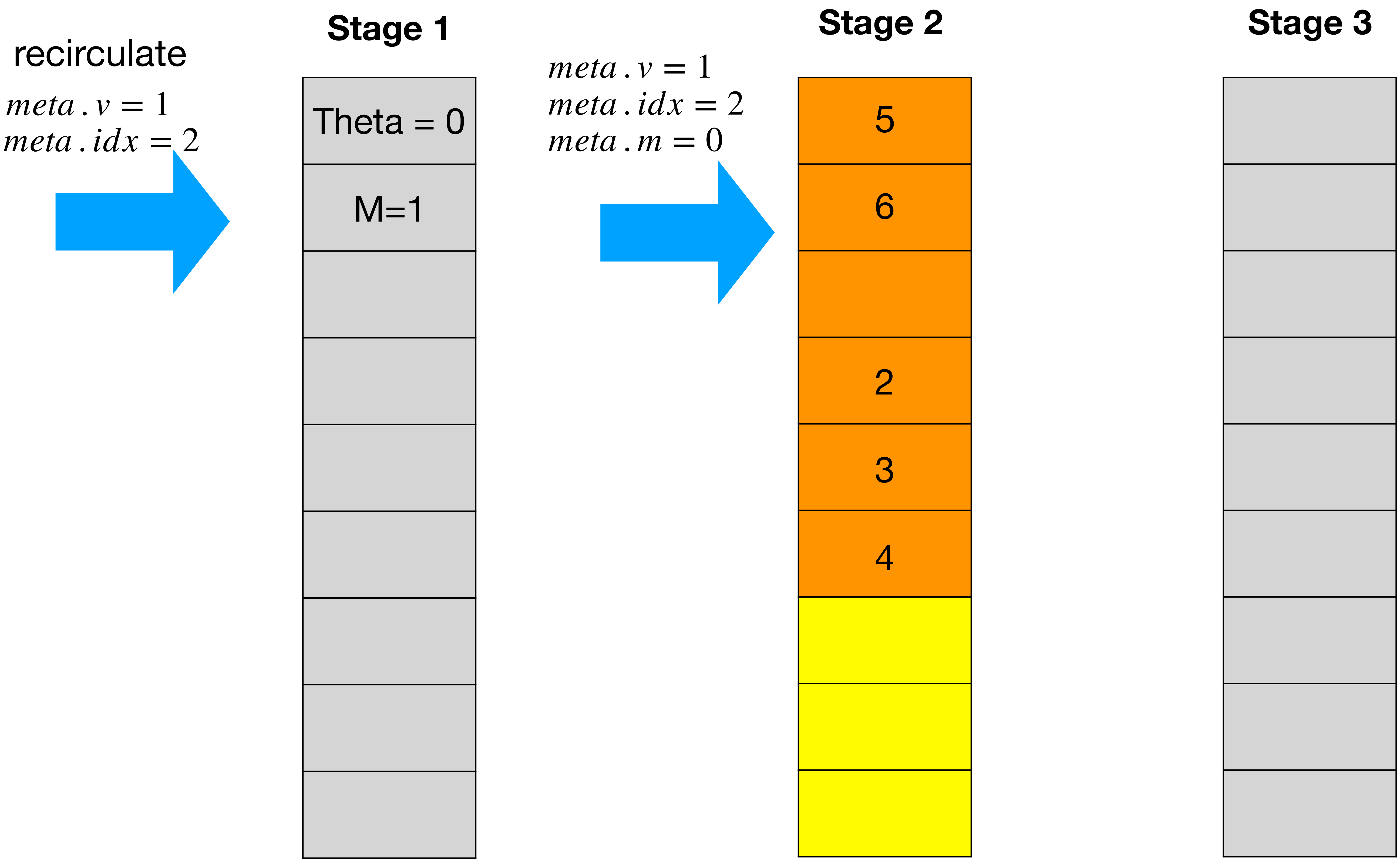
System Design



System Design

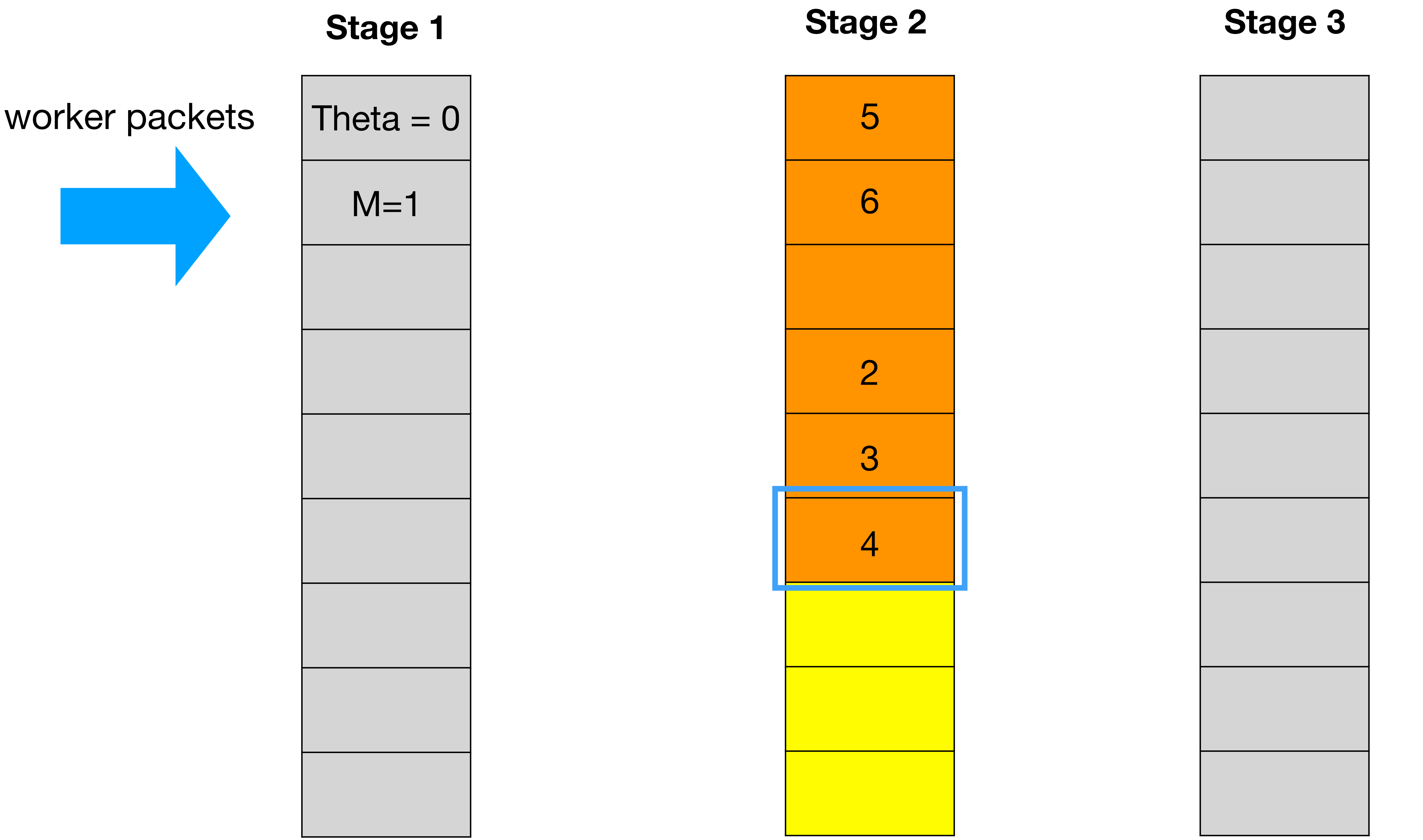


System Design



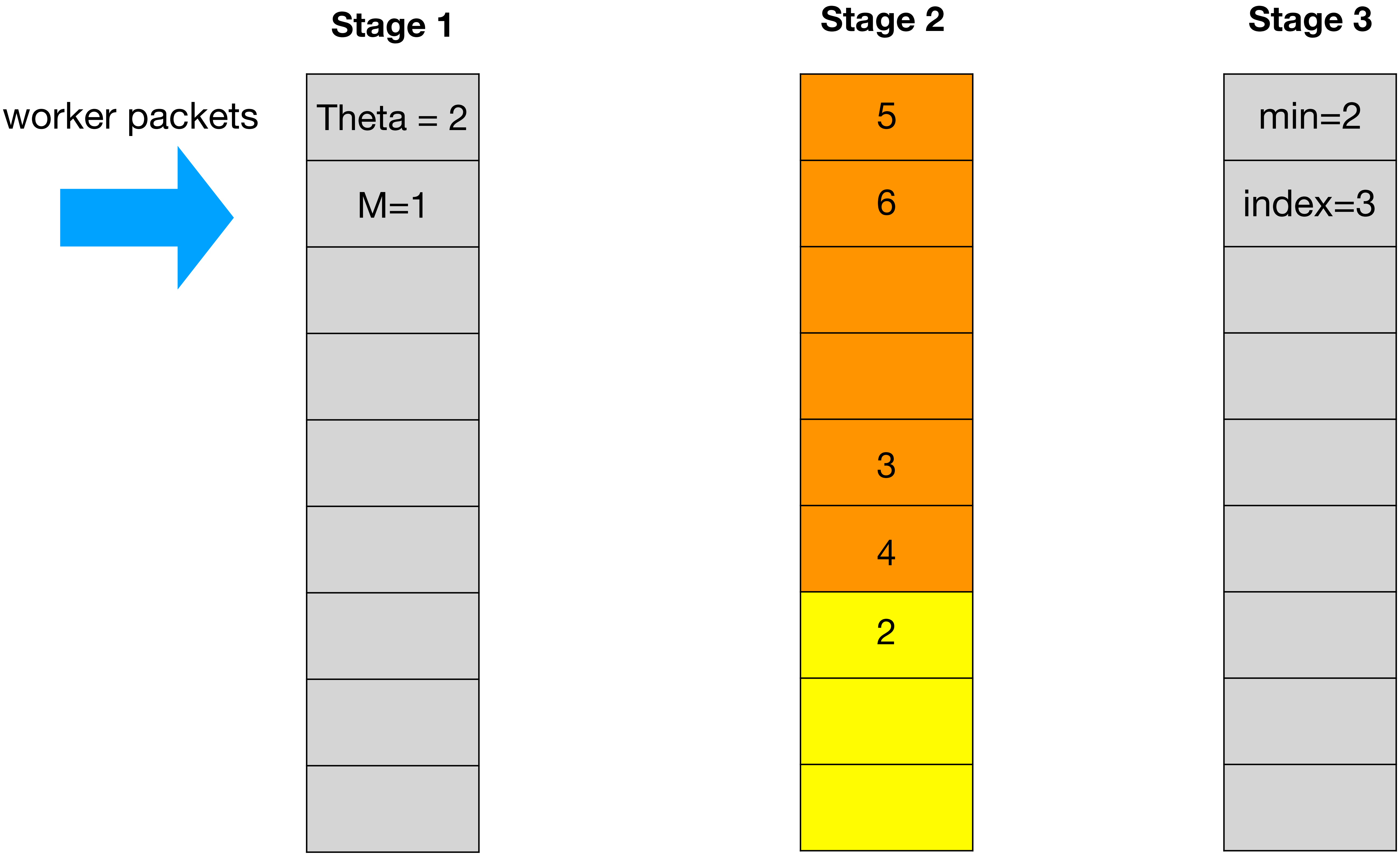
System Design

System Design



System Design

System Design



Evaluation

- **Experiment setup**
  - Three traces: traceroute-based measurements, DNS RTT measurements, and high-speed internet backbone measurements.

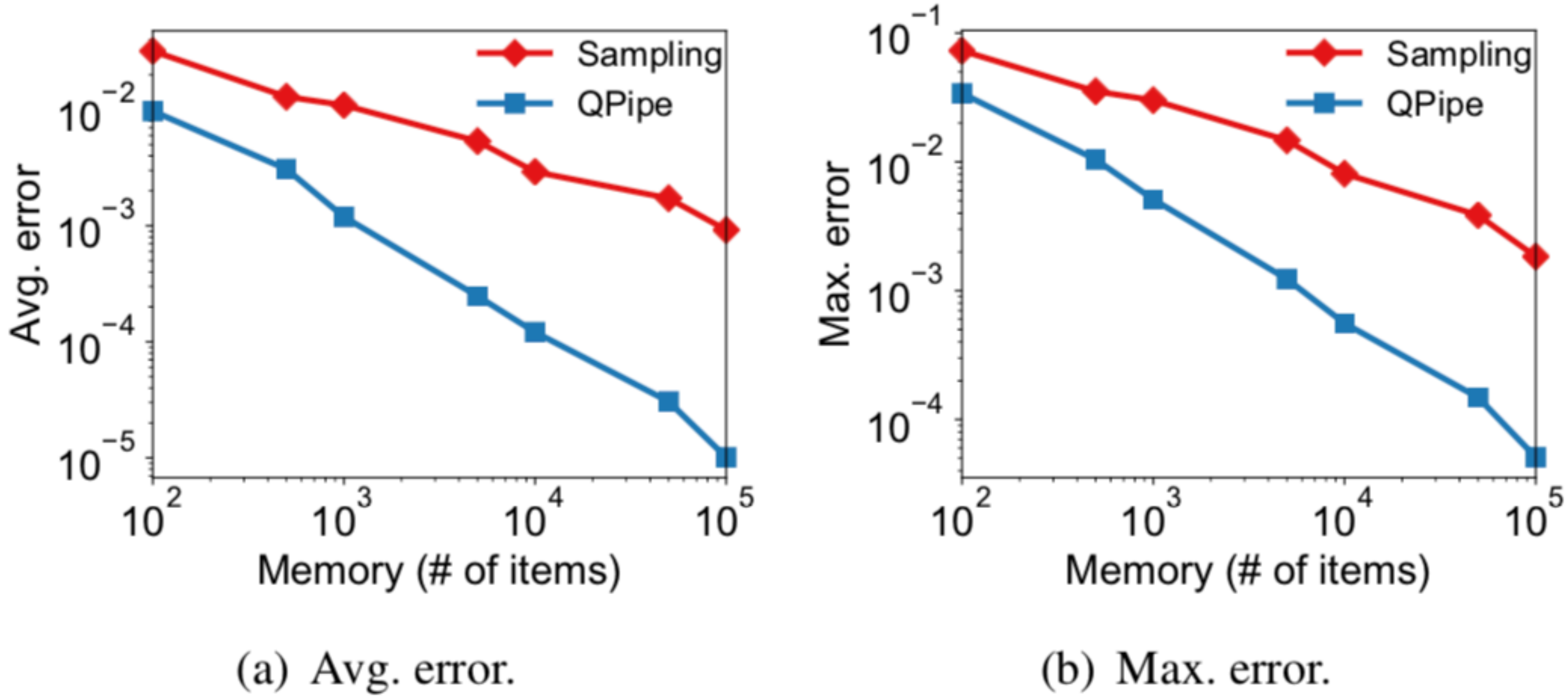
Evaluation

- **Experiment setup**
  - Three traces: traceroute-based measurements, DNS RTT measurements, and high-speed internet backbone measurements.
- **Metrics**
  - Avg. and Max. approximation error of finding quantile.
  - True positive rate (TPR) and false positive rate (FPR) of finding heavy hitters.

Evaluation

- **Experiment setup**
  - Three traces: traceroute-based measurements, DNS RTT measurements, and high-speed internet backbone measurements.
- **Metrics**
  - Avg. and Max. approximation error of finding quantile.
  - True positive rate (TPR) and false positive rate (FPR) of finding heavy hitters.
- **Comparison**
  - QPipe
  - Sampling
  - Count-min Sketch

Evaluation

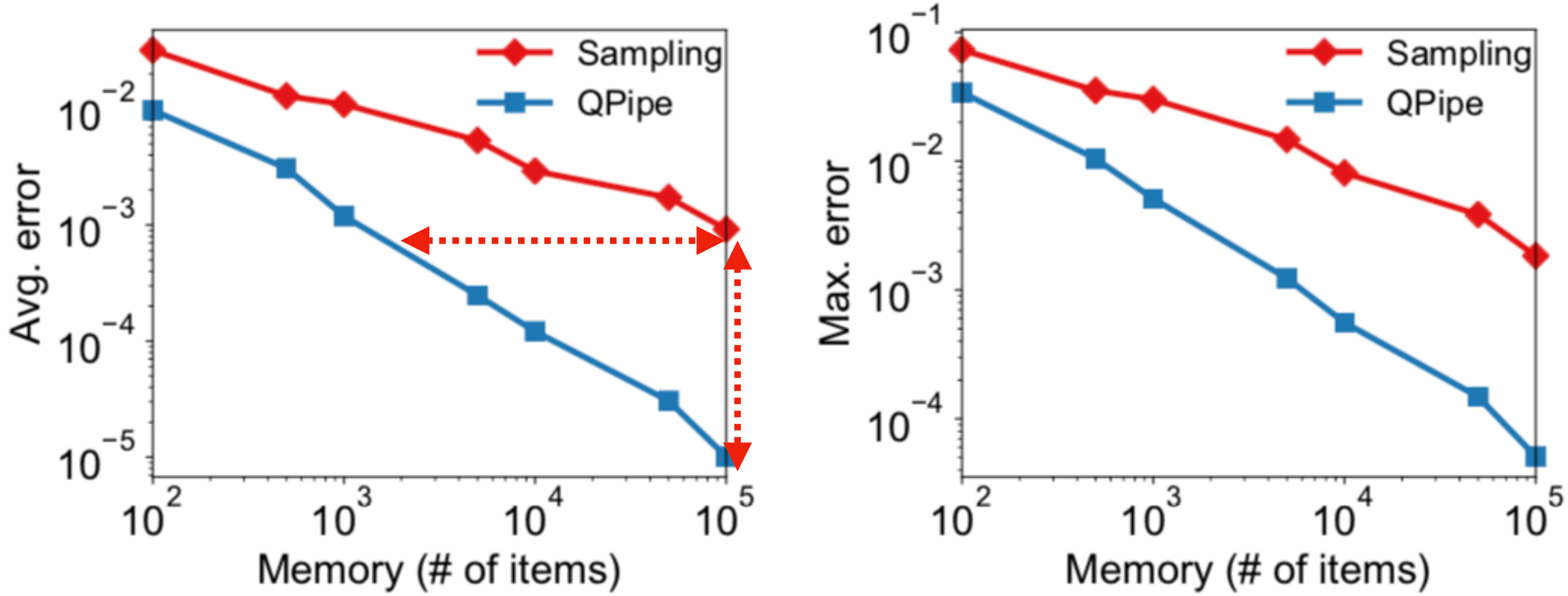


(a) Avg. error.

(b) Max. error.

Figure 5: Performance comparison of QPipe and Sampling under different memory size in trace (a) with source IP address as the key.

Evaluation



(a) Avg. error. (b) Max. error.

Figure 5: Performance comparison of QPipe and Sampling under different memory size in trace (a) with source IP address as the key.

90x improvement!

Evaluation

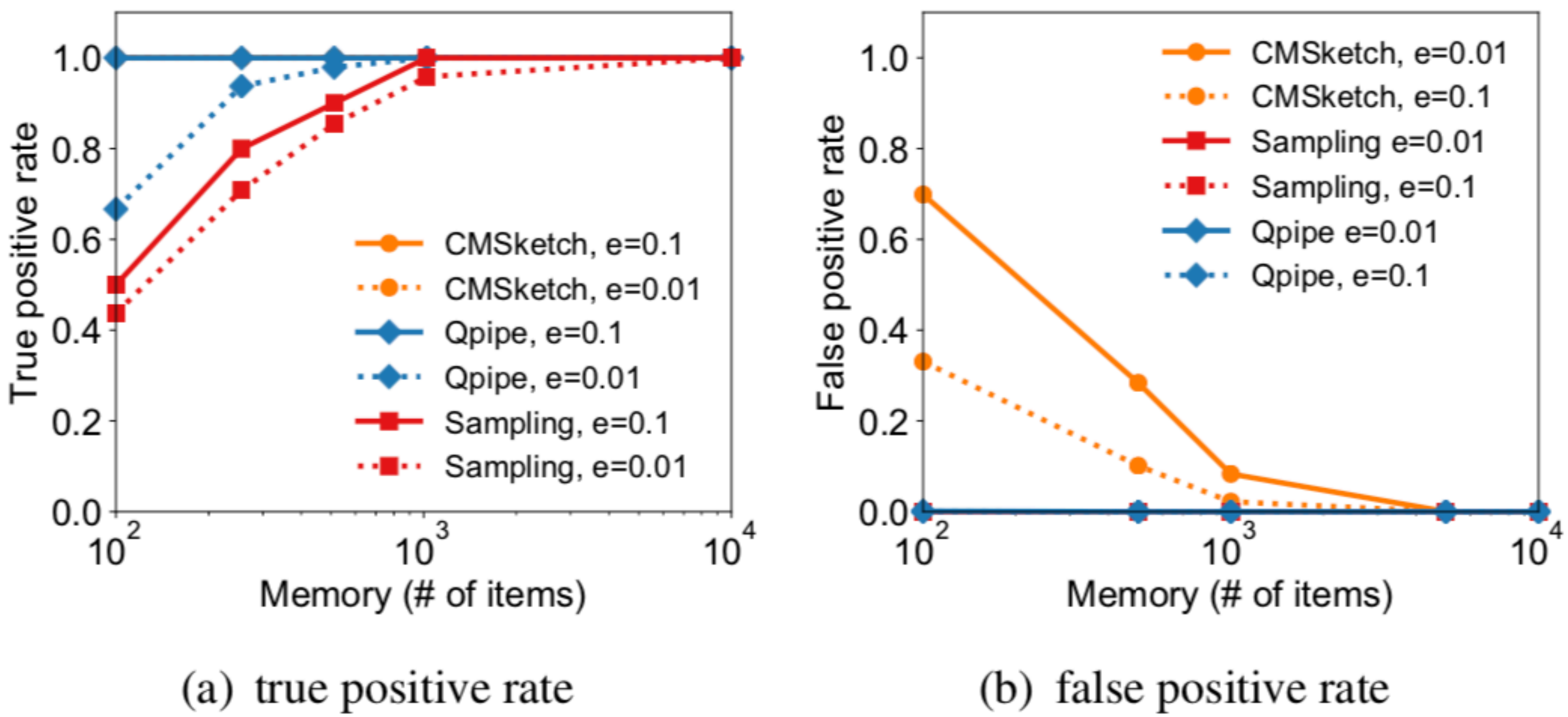
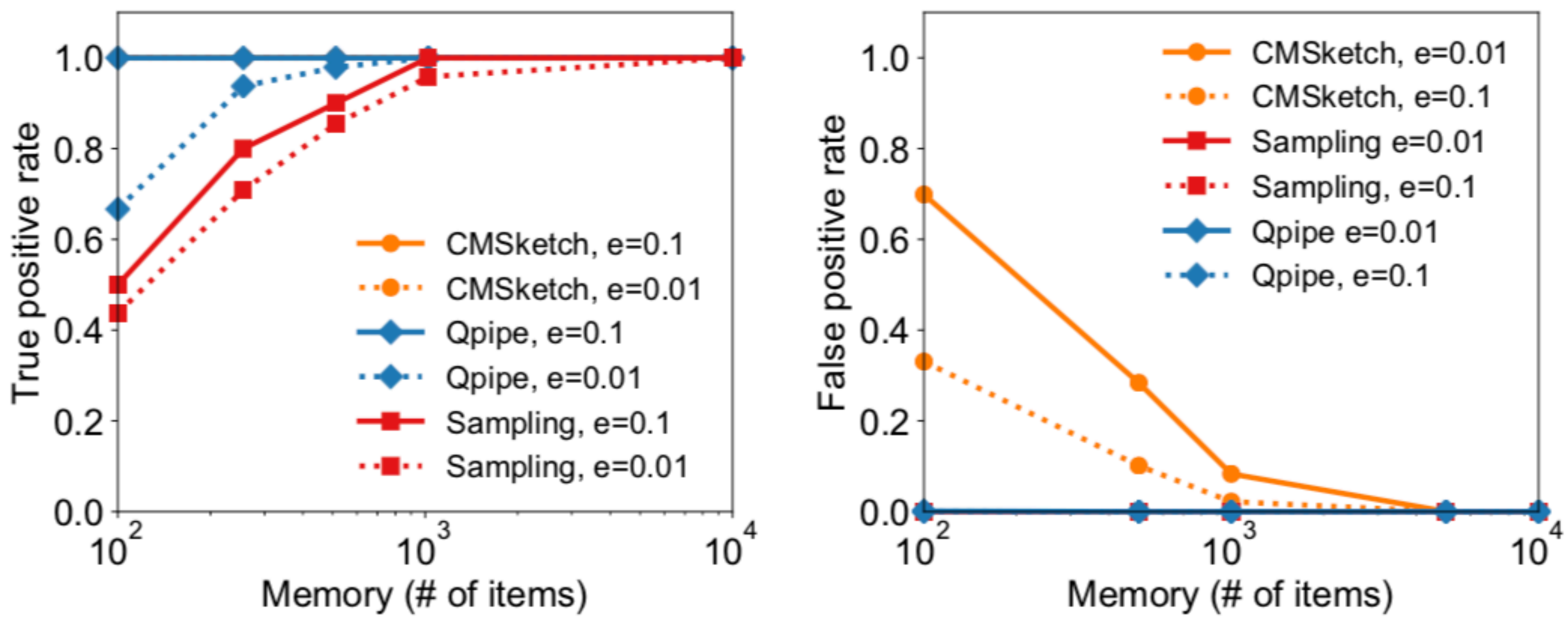


Figure 7: Performance comparison of QPipe, Sampling and Count-Min Sketch under different memory size for finding heavy hitters in trace (c) with source IP address as the key.

Evaluation



(a) true positive rate (b) false positive rate

Figure 7: Performance comparison of QPipe, Sampling and Count-Min Sketch under different memory size for finding heavy hitters in trace (c) with source IP address as the key.

Low false positive rate!

Conclusion

- We present QPipe, to the best of our knowledge, the first quantiles sketching algorithm implemented in the data plane.

- We show 90x improvement in precision under a fixed memory budget compared with sampling baseline.

Conclusion

- We present QPipe, to the best of our knowledge, the first quantiles sketching algorithm implemented in the data plane.

- We show 90x improvement in precision under a fixed memory budget compared with sampling baseline.

Takeaway

1. Report **quantiles** in the data plane
2. Employing **“worker packets”**

Code available at https://github.com/netx-repo/QPipe

Thank you!