FAST'19

Write-Optimized Dynamic Hashing for Persistent Memory

Moohyeon Nam[†], *Hokeun Cha*[δ], *Young - ri Choi*[†], *Sam H. Noh*[†], *Beomseok Nam*[δ]

UNIST (*Ulsan National Institute of Science and Technology*)[†]
Sungkyunkwan University (*SKKU*)[δ]

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

SUNG KYUN KWAN UNIVERSITY

Outline

Background

- Static Hashing
- Extendible Hashing
- Persistent Memory

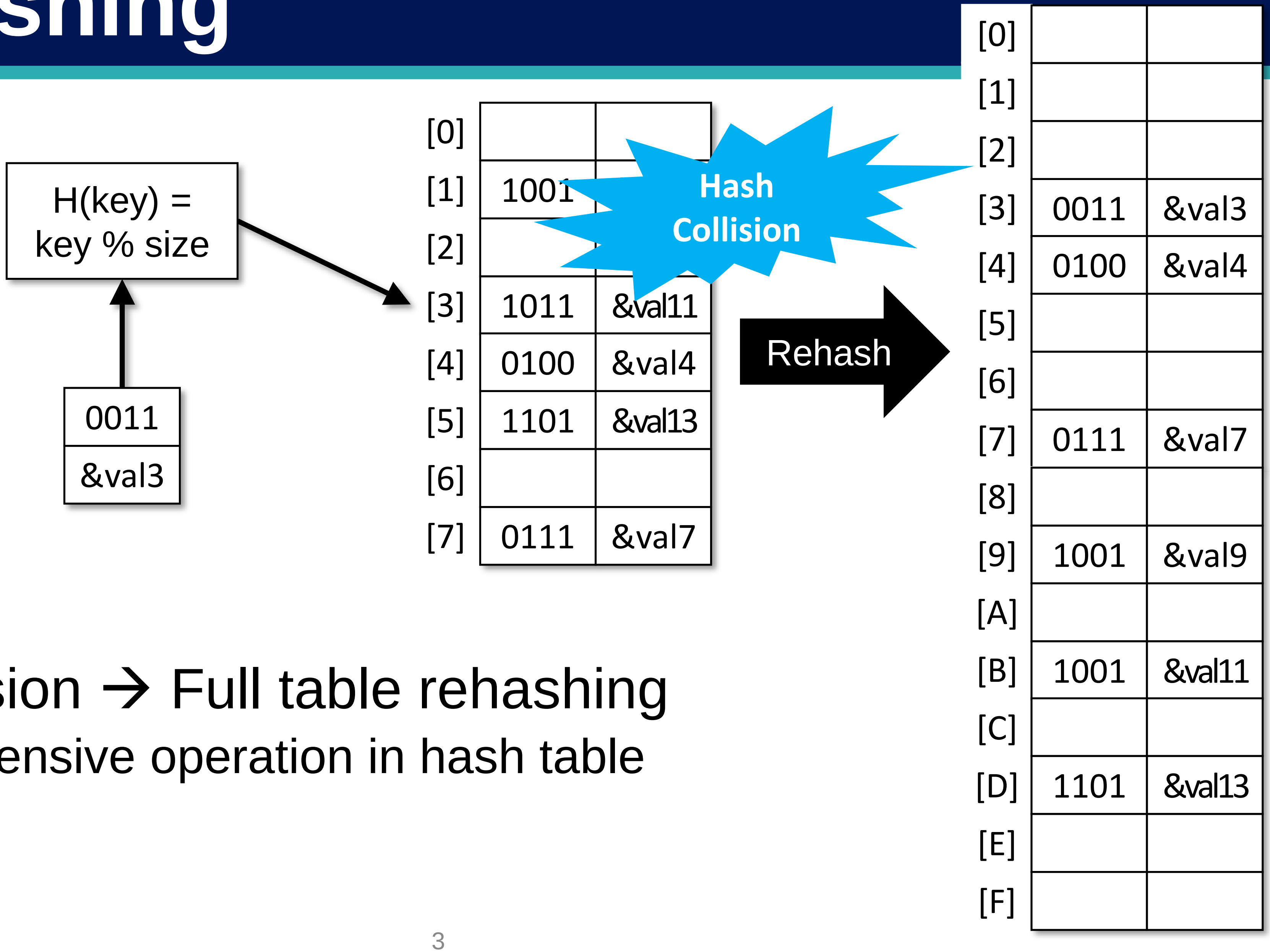
Cacheline-Conscious Extendible Hashing

- Challenges and Contributions
- 3-Level Structure of CCEH
- Failure-atomic Directory Update

Evaluation

Conclusion

Background:
Static Hashing

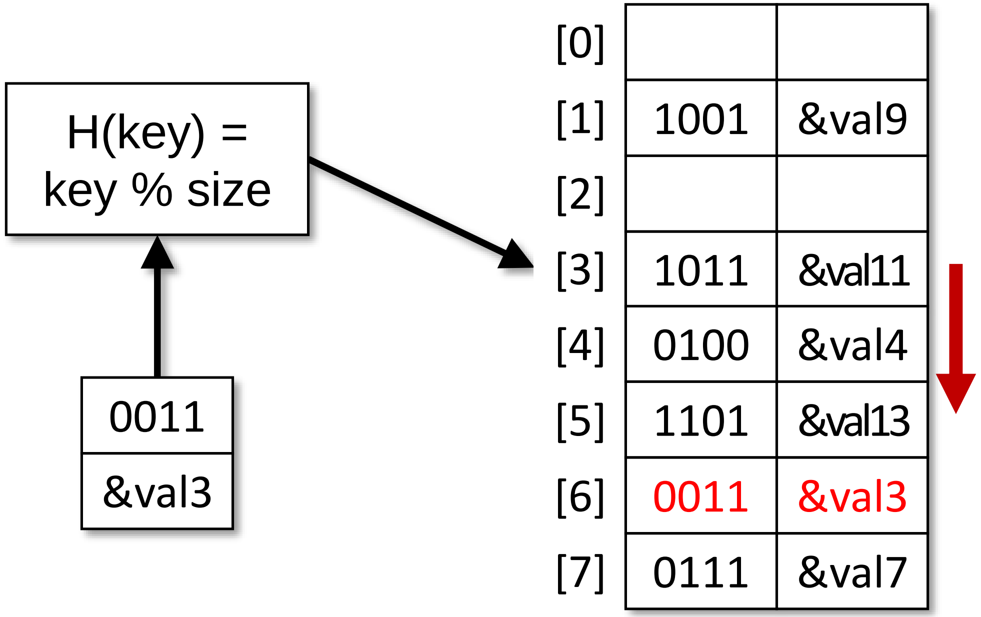


[0]		
[1]	1001	
[2]		
[3]	1011	&val11
[4]	0100	&val4
[5]	1101	&val13
[6]		
[7]	0111	&val7

[0]		
[1]		
[2]		
[3]	0011	&val3
[4]	0100	&val4
[5]		
[6]		
[7]	0111	&val7
[8]		
[9]	1001	&val9
[A]		
[B]	1001	&val11
[C]		
[D]	1101	&val13
[E]		
[F]		

- Hash key collision → Full table rehashing
  - The most expensive operation in hash table

Background:
Static Hashing

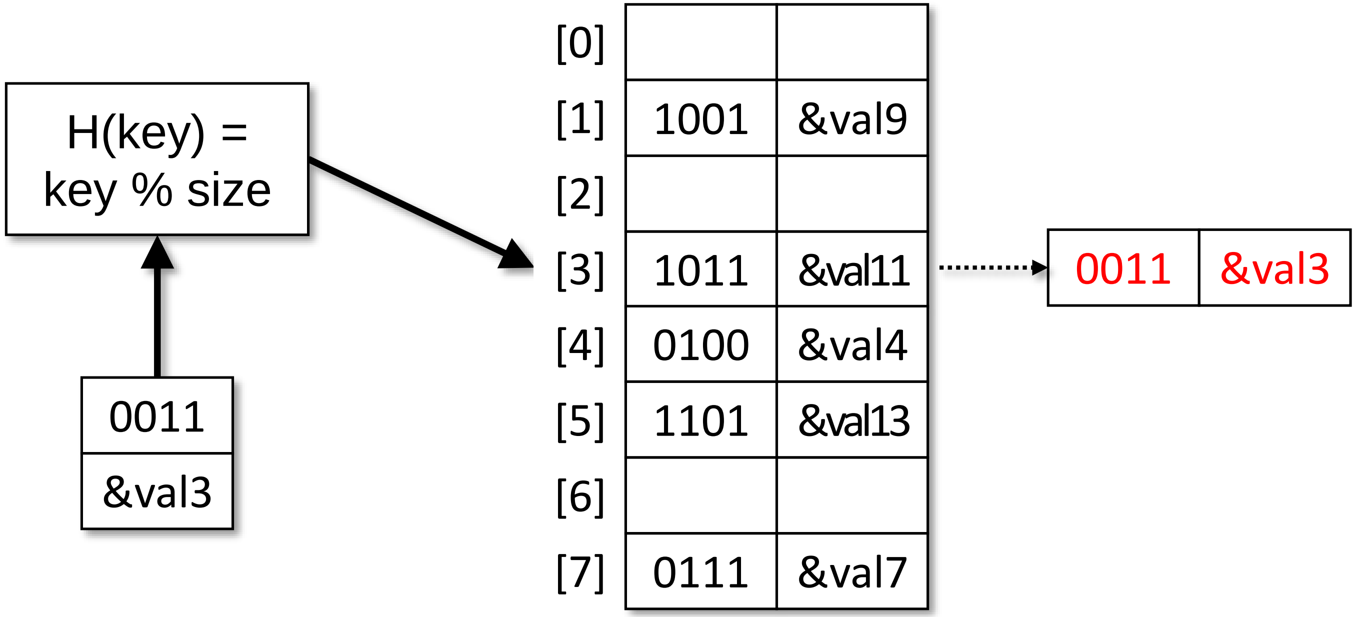


[0]		
[1]	1001	&val9
[2]		
[3]	1011	&val11
[4]	0100	&val4
[5]	1101	&val13
[6]	0011	&val3
[7]	0111	&val7

Linear Probing

- To avoid full table rehashing:
  - Linear probing
  - Chaining
  - Double hashing such as Cuckoo hashing

Background:
Static Hashing



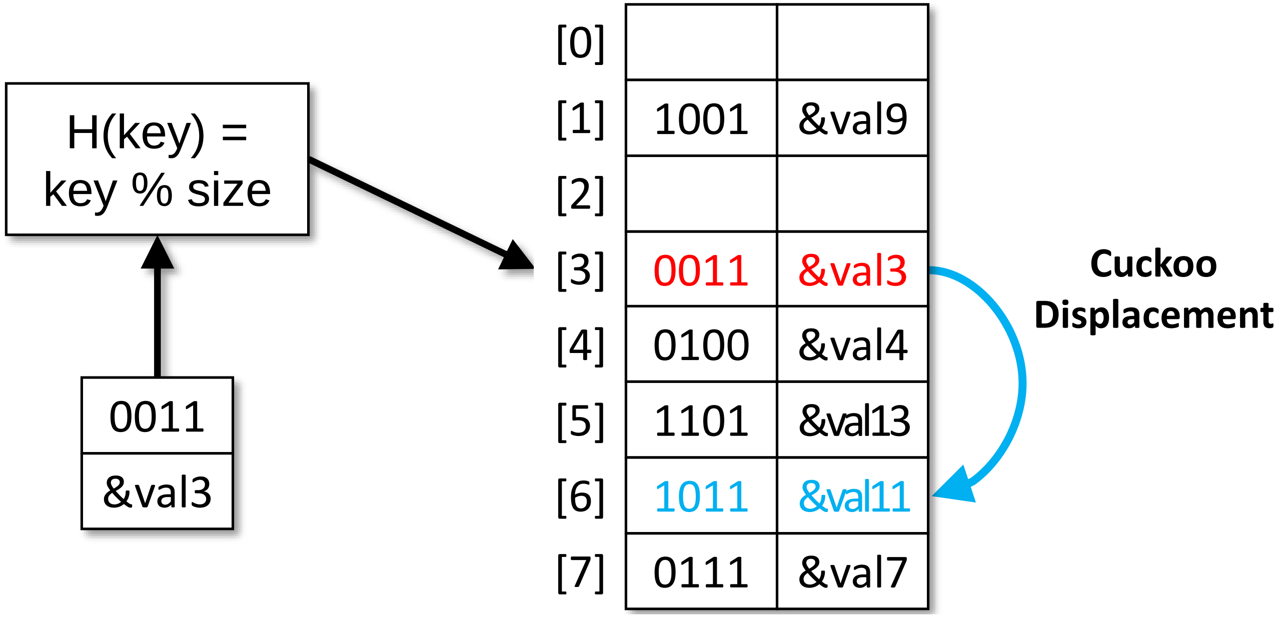
H(key) =
key % size

[0]		
[1]	1001	&val9
[2]		
[3]	1011	&val11
[4]	0100	&val4
[5]	1101	&val13
[6]		
[7]	0111	&val7

Chaining

- To avoid full table rehashing:
  - Linear probing
  - Chaining
  - Double hashing such as Cuckoo hashing

Background:
Static Hashing



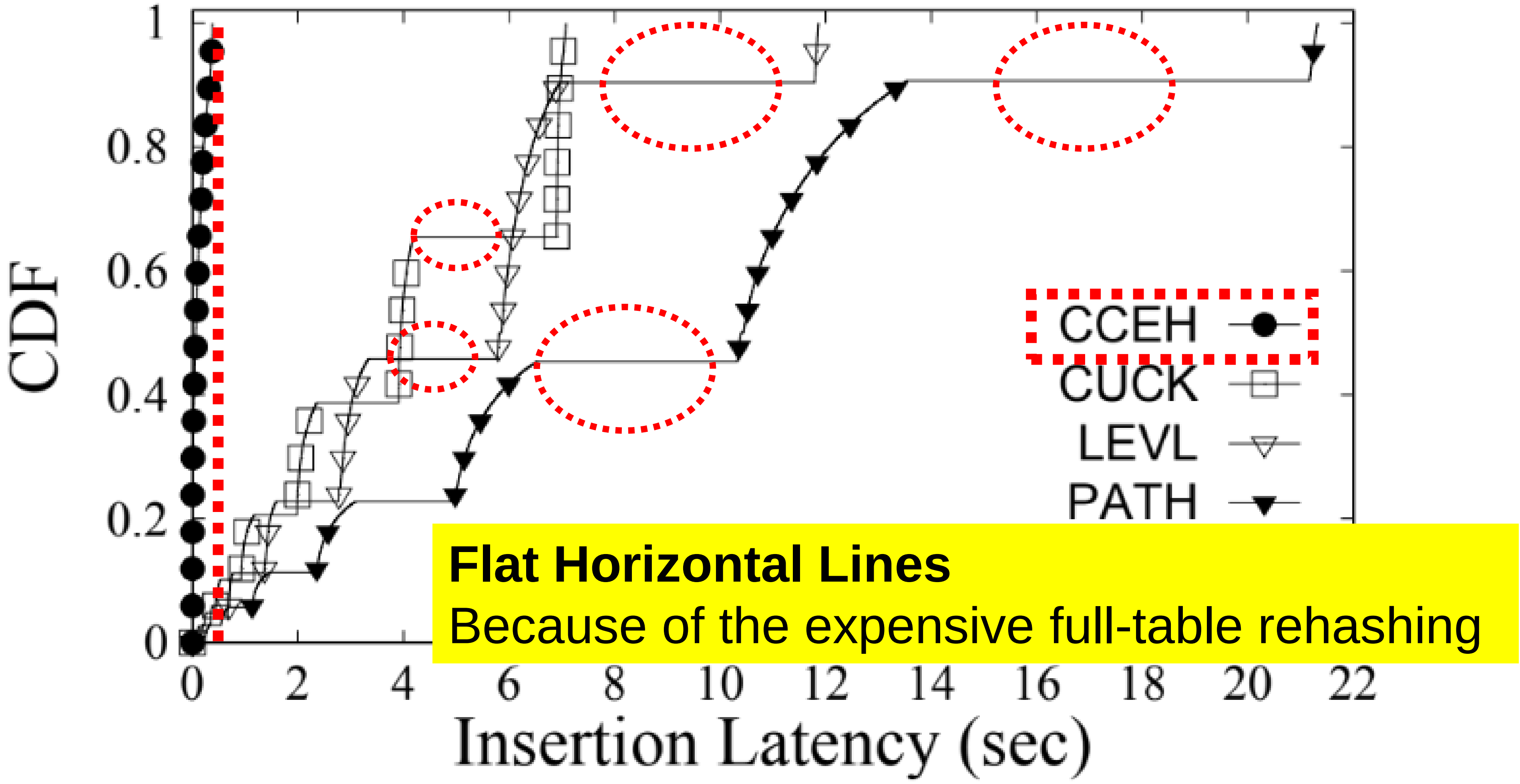
[0]		
[1]	1001	&val9
[2]		
[3]	0011	&val3
[4]	0100	&val4
[5]	1101	&val13
[6]	1011	&val11
[7]	0111	&val7

Cuckoo Displacement

Cuckoo Hashing

- To avoid full table rehashing:
  - Linear probing
  - Chaining
  - Double hashing such as Cuckoo hashing

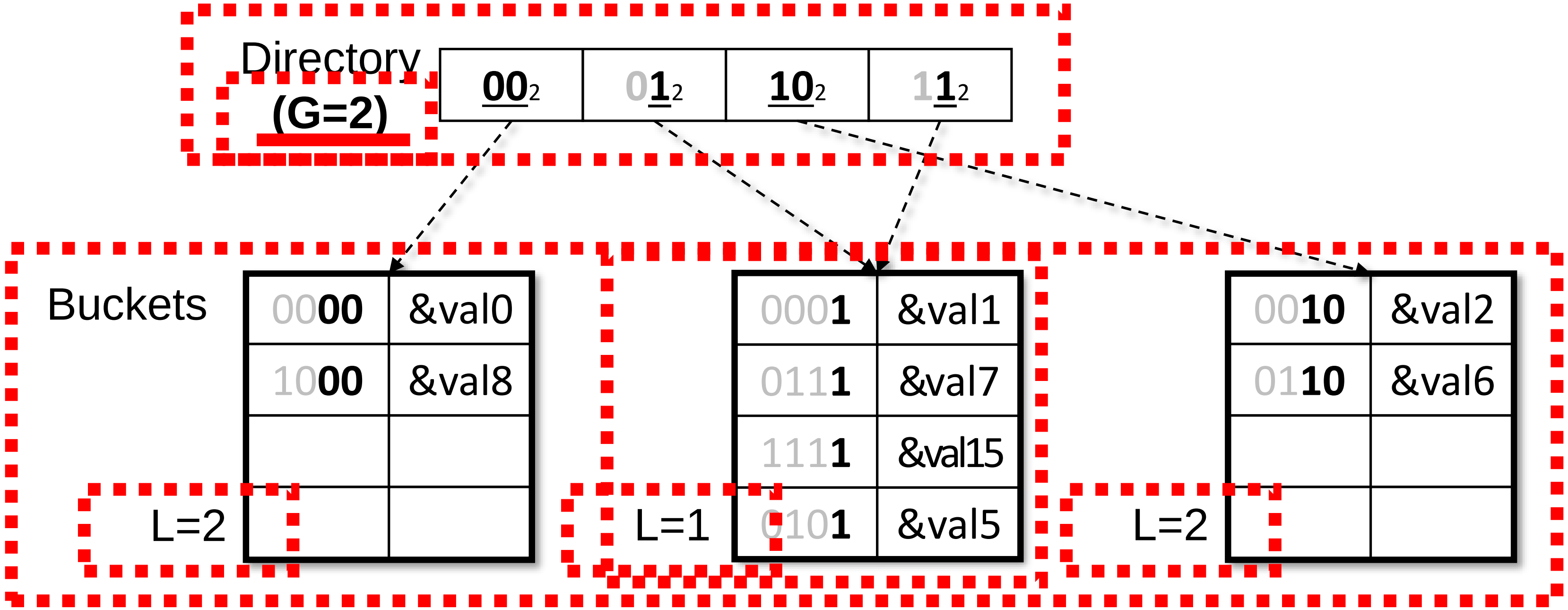
Insertion Latency CDF



Flat Horizontal Lines
Because of the expensive full-table rehashing

Background:
Disk-based Extendible Hashing

Hash Function:
H(key) = key % 2^G



Directory
(G=2)

00_2	01_2	10_2	11_2

Buckets

0000	&val0
1000	&val8

L=2

0001	&val1
0111	&val7
1111	&val15
0101	&val5

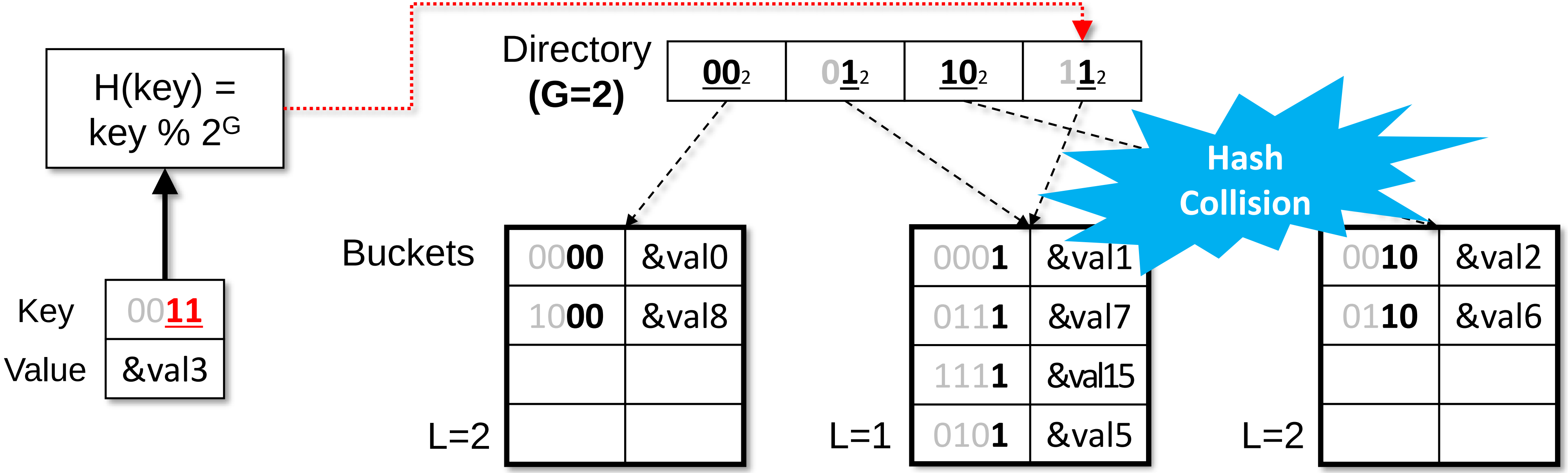
L=1

0010	&val2
0110	&val6

L=2

- Dynamically splits one bucket or merges two buckets at a time

Background:
Extendible Hashing – Insertion



H(key) = key % 2^G

Key	0011
Value	&val3

Directory (G=2)

00_2	01_2	10_2	11_2

Buckets

L=2

0000	&val0
1000	&val8

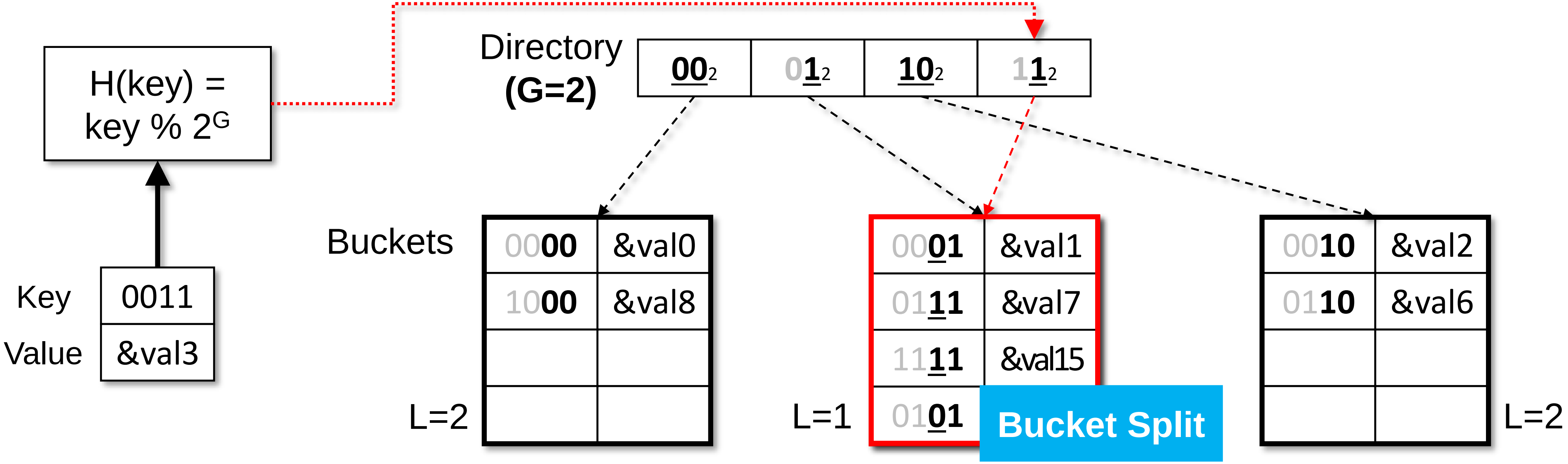
L=1

0001	&val1
0111	&val7
1111	&val15
0101	&val5

L=2

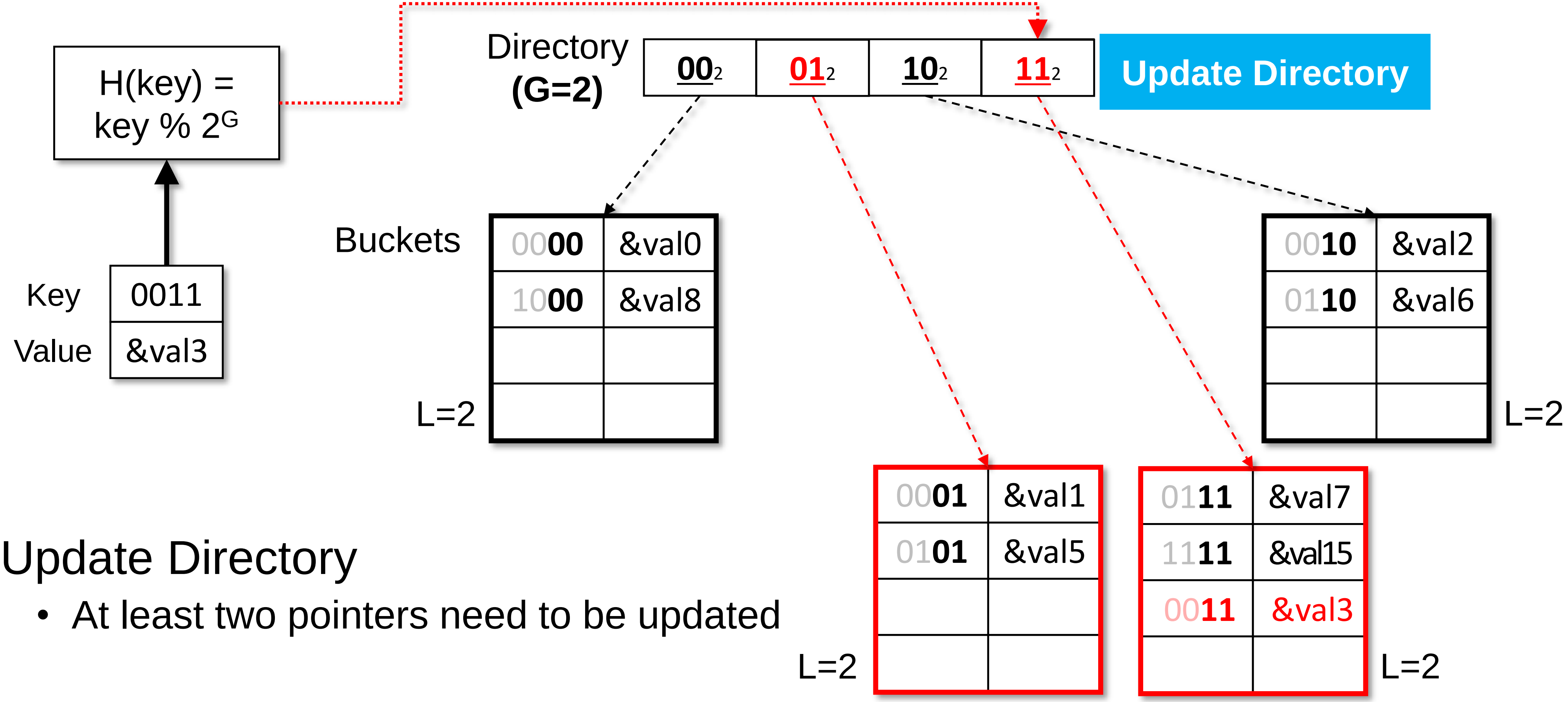
0010	&val2
0110	&val6

Background:
Extendible Hashing – Bucket Split



- Only overflown bucket is modified

Background:
Extendible Hashing – Bucket Split



Key	0011
Value	&val3

0000	&val0
1000	&val8

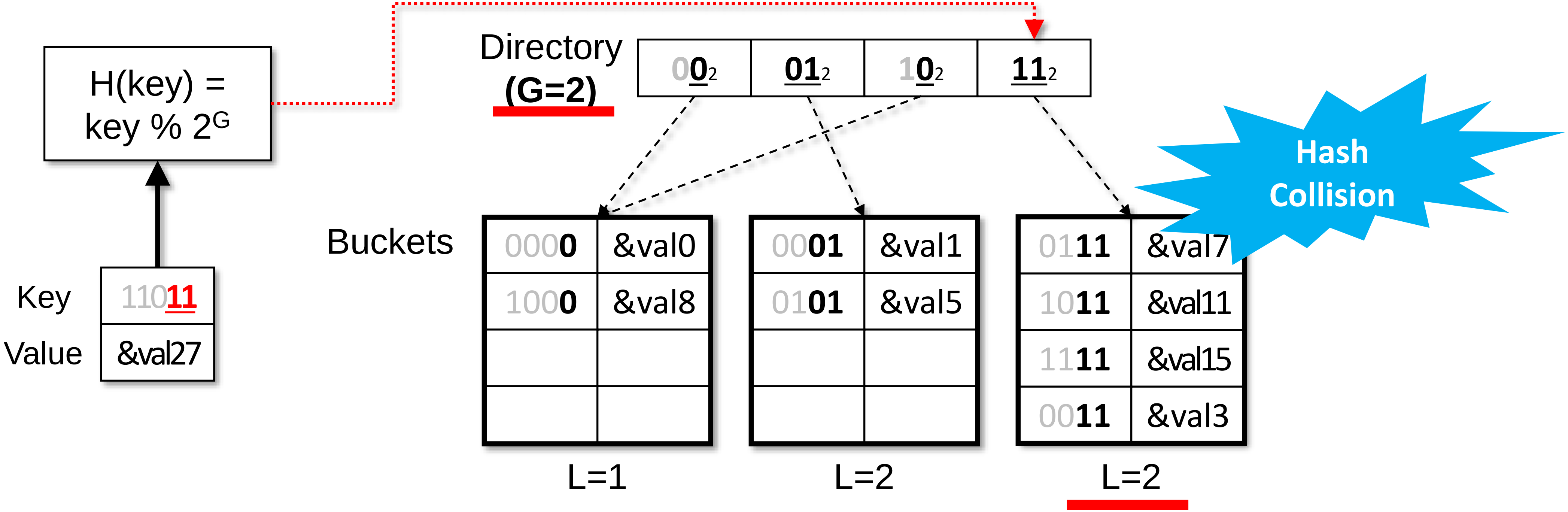
0001	&val1
0101	&val5

0111	&val7
1111	&val15
0011	&val3

0010	&val2
0110	&val6

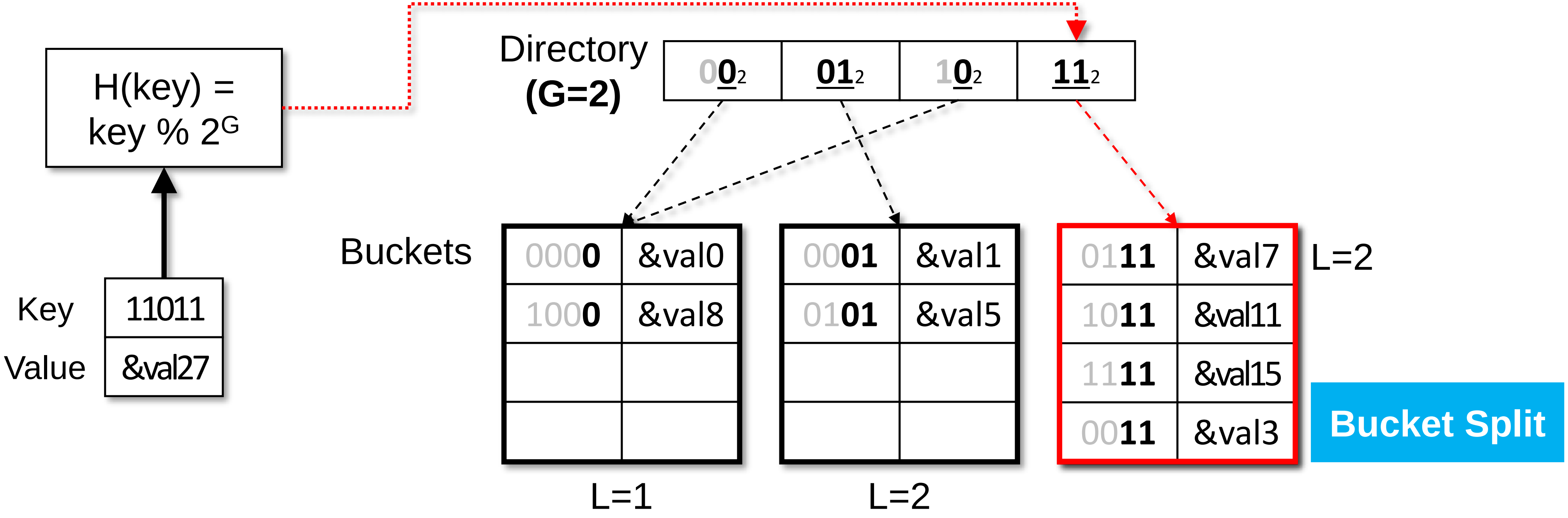
- Update Directory
  - At least two pointers need to be updated

Background:
Extendible Hashing – Directory doubling



- If a single pointer points to overflow bucket
  → Directory Doubling

Background:
Extendible Hashing – Directory doubling



0000	&val0
1000	&val8

0001	&val1
0101	&val5

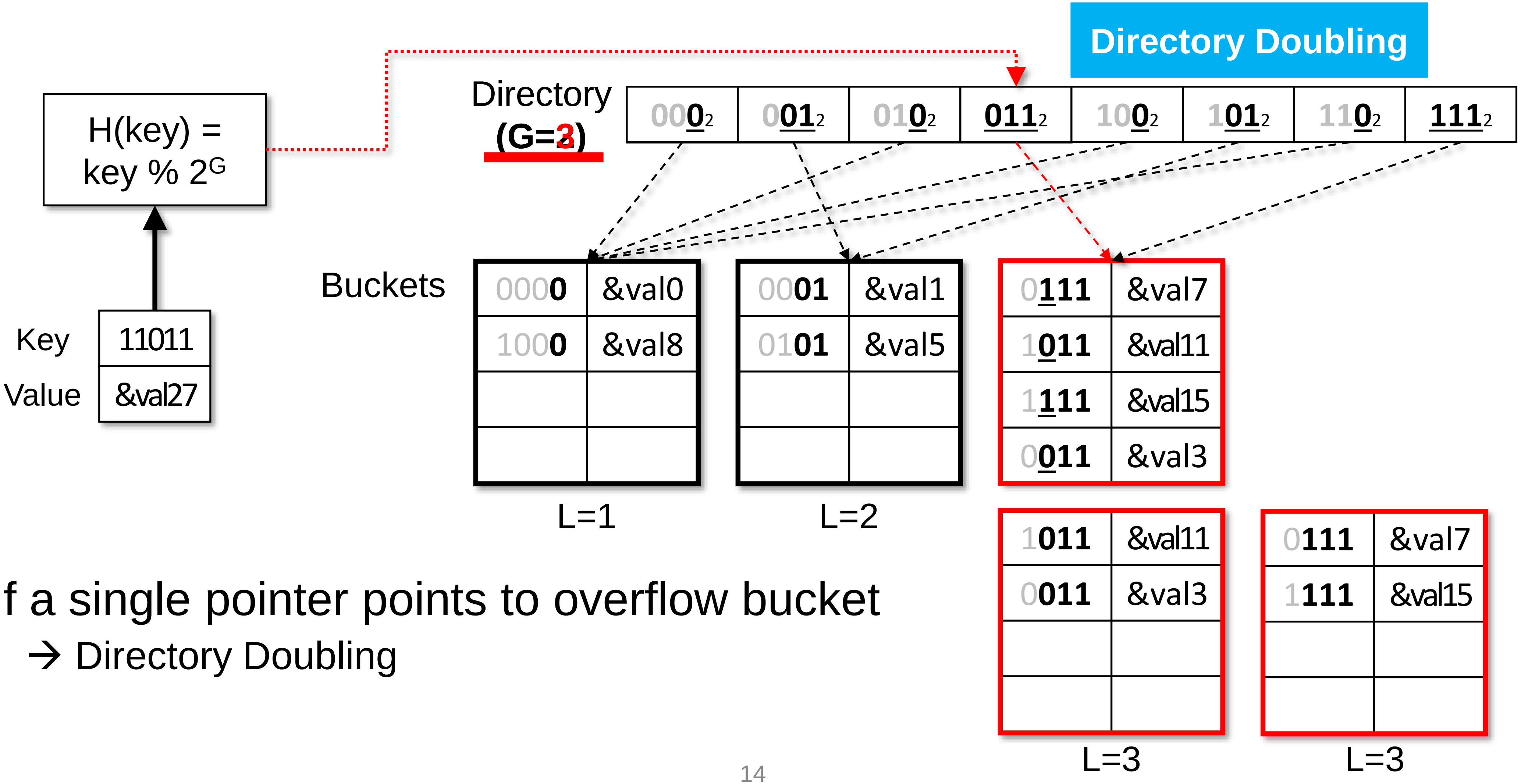
0111	&val7
1011	&val11
1111	&val15
0011	&val3

- If a single pointer points to overflow bucket

  → Directory Doubling

Background:

Extendible Hashing – Directory doubling



Key	11011
Value	&val27

0000	&val0
1000	&val8

L=1

0001	&val1
0101	&val5

L=2

0111	&val7
1011	&val11
1111	&val15
0011	&val3

1011	&val11
0011	&val3

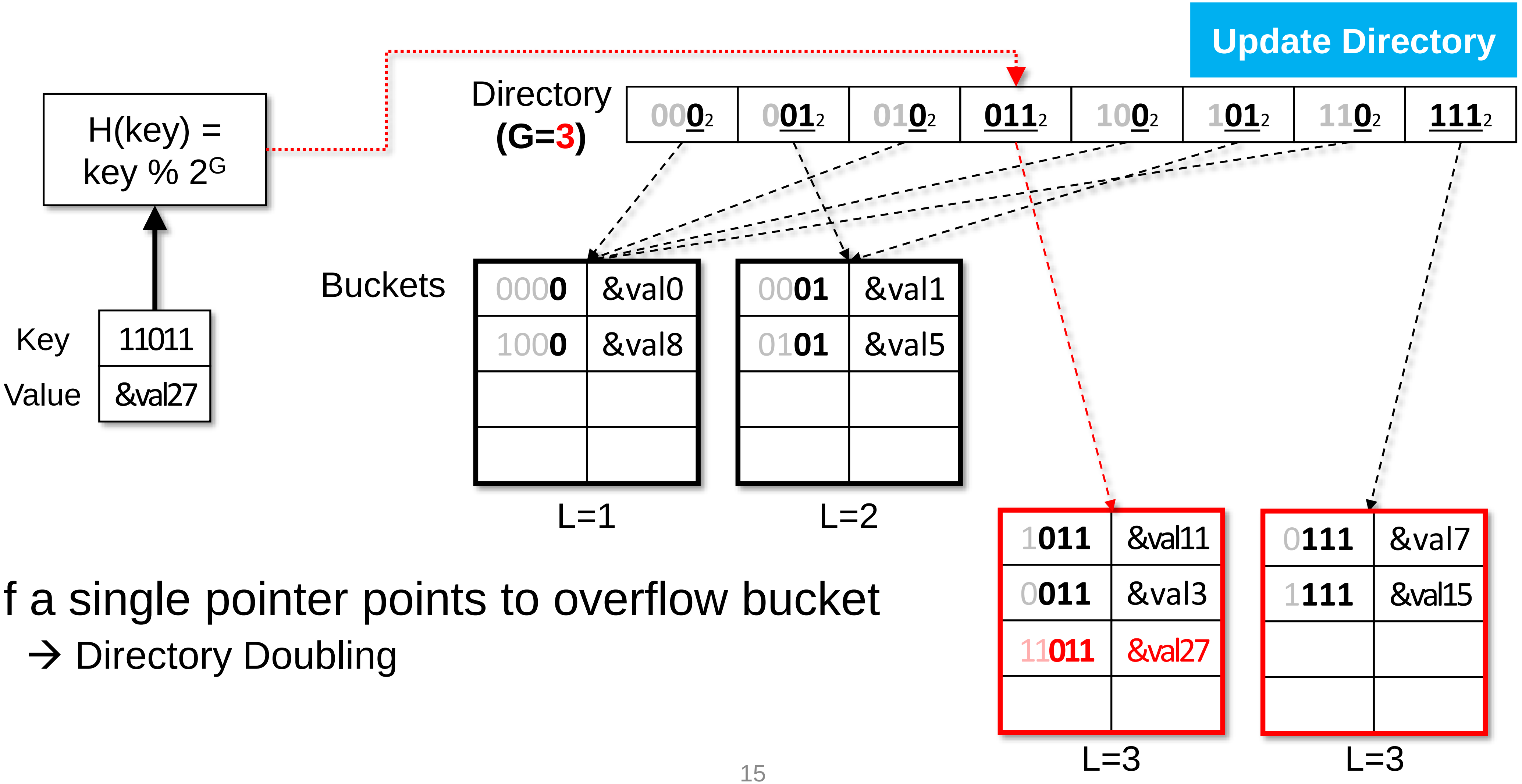
L=3

0111	&val7
1111	&val15

L=3

- If a single pointer points to overflow bucket
  → Directory Doubling

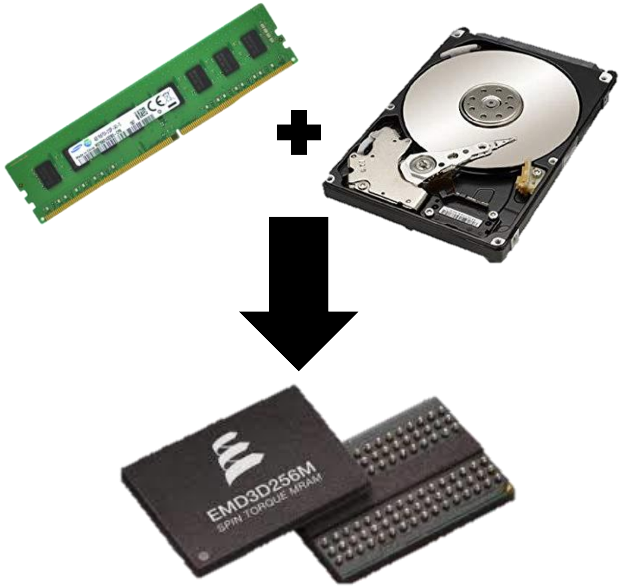
Background:
Extendible Hashing – Directory doubling



- If a single pointer points to overflow bucket

  → Directory Doubling

Persistent Memory



Characteristics

- High performance – Comparable to DRAM
- Byte-addressability – As DRAM
- Persistence – As storage devices (HDD/SSD)

Challenges

- Atomic unit of writes → 8-bytes
- Data transfer unit between CPU cache and PM → 64 byte cacheline
- Order of memory writes is not guaranteed

Outline

~~Background~~

- ~~Static Hashing~~
- ~~Extendible Hashing~~
- ~~Persistent Memory~~

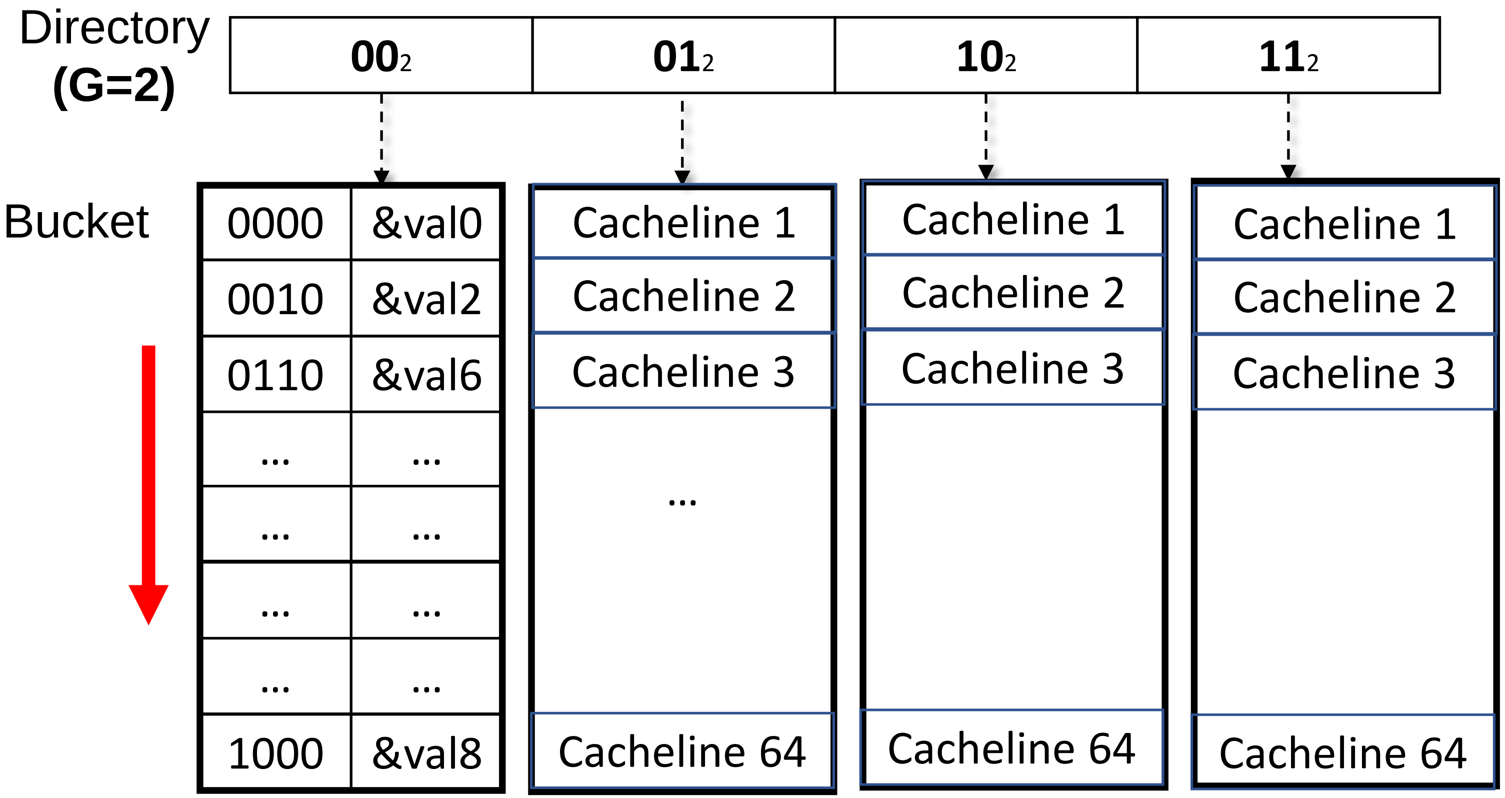
Cacheline-Conscious Extendible Hashing

- Challenges and Contributions
- 3-Level Structure of CCEH
- Failure-atomic Directory Update

Evaluation

Conclusion

Challenge in In-Memory Extendible Hashing



Directory
(G=2)

00_2	**01_2**	**10_2**	**11_2**

Bucket

0000	&val0
0010	&val2
0110	&val6
…	…
…	…
…	…
…	…
1000	&val8

Cacheline 1
Cacheline 2
Cacheline 3
…
Cacheline 64

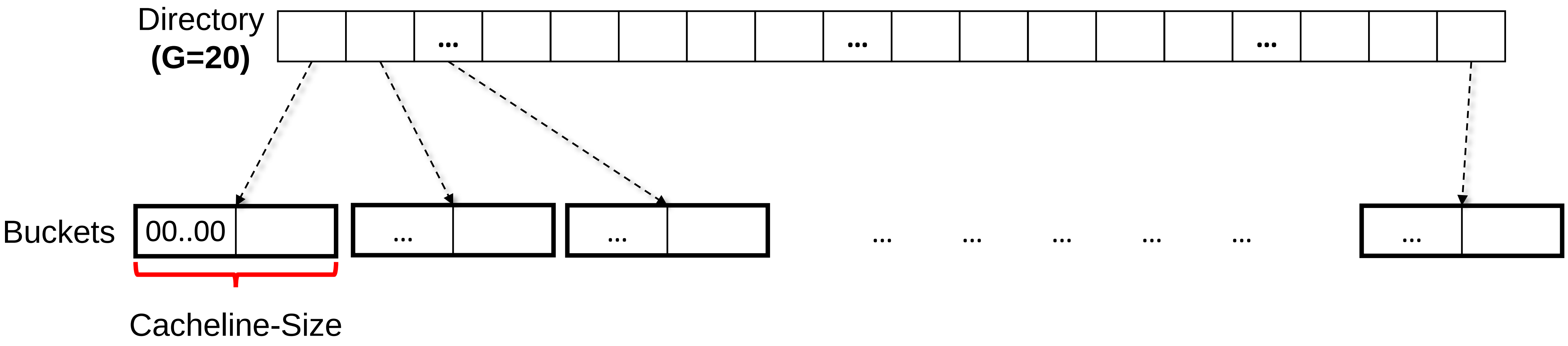
Cacheline 1
Cacheline 2
Cacheline 3
Cacheline 64

Cacheline 1
Cacheline 2
Cacheline 3
Cacheline 64

Problems

- Page-sized bucket → 64 cacheline accesses per bucket

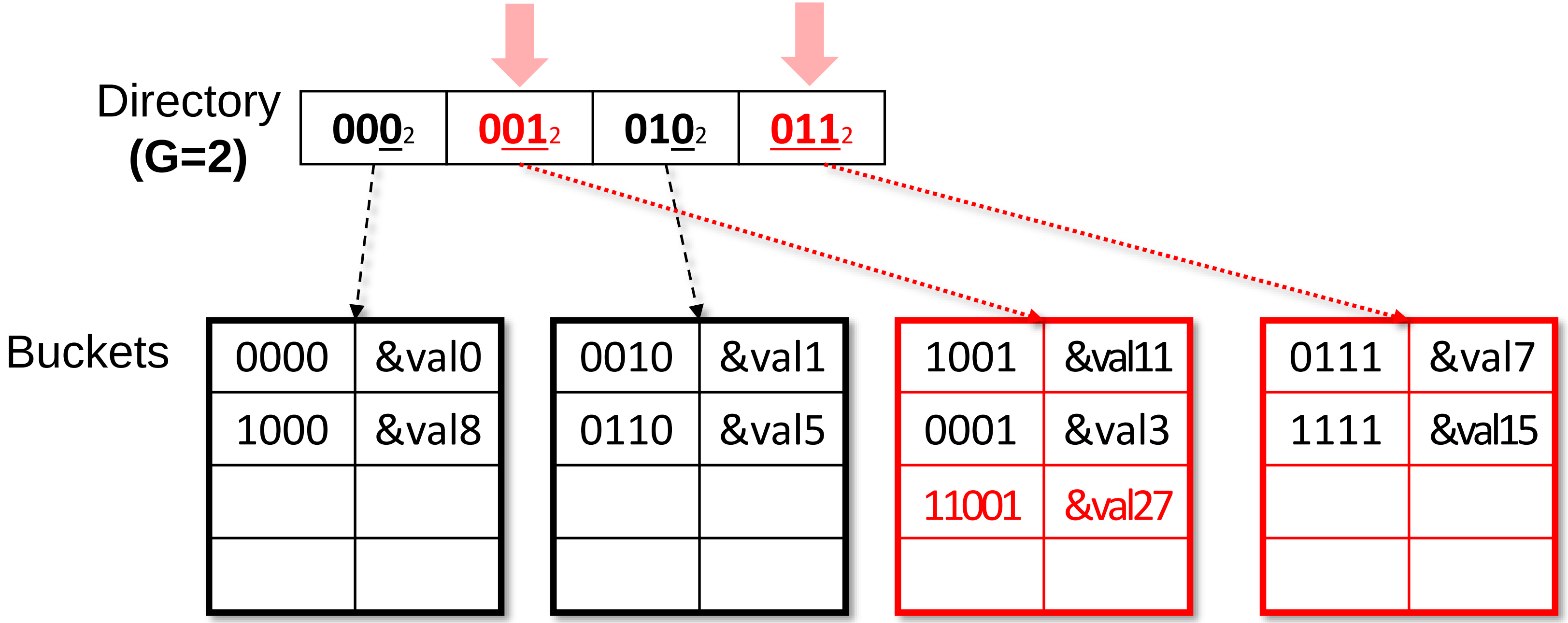
Challenge in In-Memory Extendible Hashing



Problems

Cacheline-sized small bucket → a large directory
(8 byte pointer per cacheline)

Challenge in Extendible Hashing on PM



Directory
(G=2)

000_2	001_2	010_2	011_2

Buckets

0000	&val0
1000	&val8

0010	&val1
0110	&val5

1001	&val11
0001	&val3
11001	&val27

0111	&val7
1111	&val15

Problems

Split operation updates multiple pointers → Not Failure-Atomic

Contributions

3-Level Structure

→ Introduces an intermediate level, ***Segment***

→ Lookup via only ***two cacheline accesses***

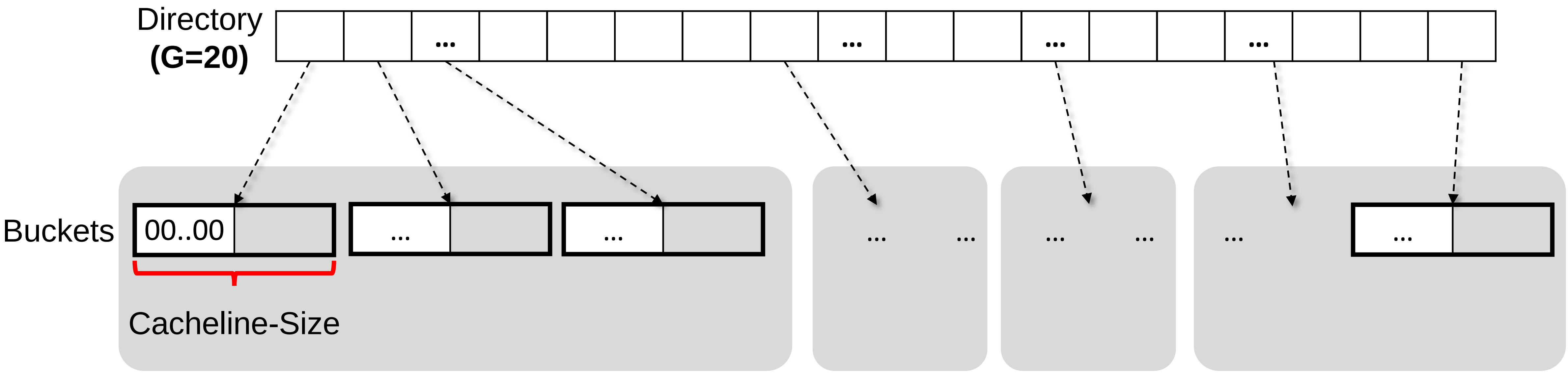
Failure-atomic Directory Updates

→ Introduces ***the split buddy tree*** to **manage split history**

Failure-atomic Segment Split

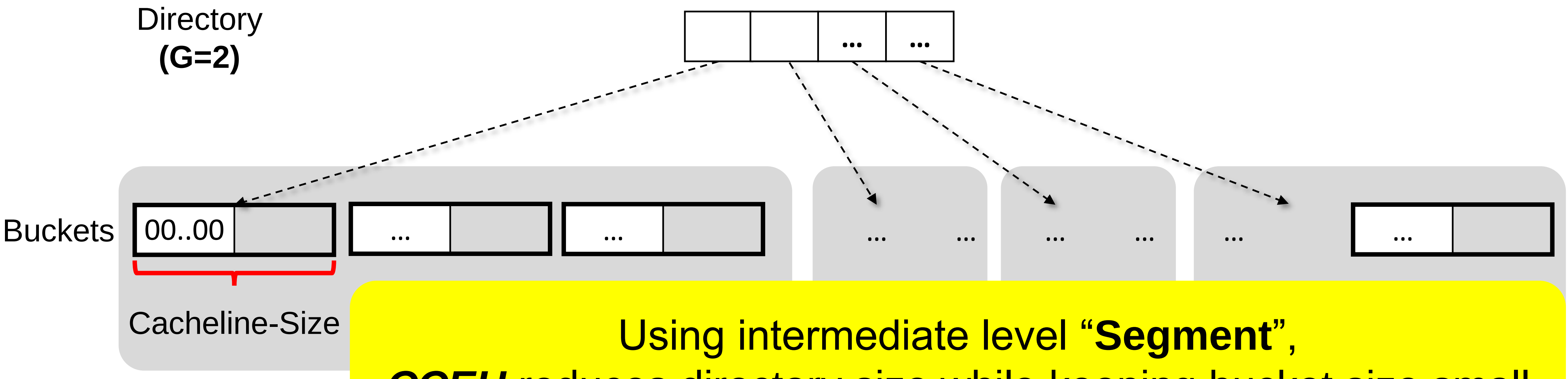
→ ***Lazy deletion*** scheme to **minimize dirty writes**

Segment: Intemediate Level



- A group of multiple cacheline-sized buckets = Segment

Segment: Intemediate Level

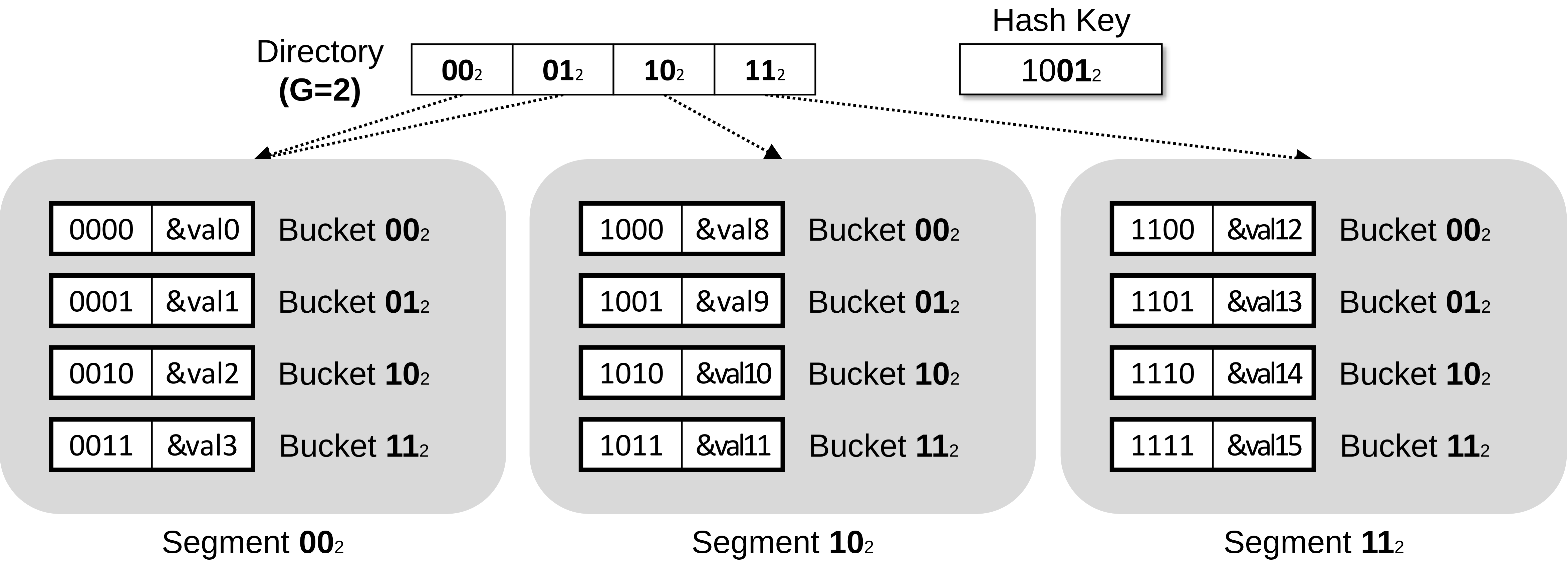


Using intermediate level “**Segment**”,
CCEH reduces directory size while keeping bucket size small

3-Level Structure

Directory → Segment → Cacheline-sized Bucket

Minimize Cacheline Accesses in Segment



0000	&val0	Bucket **00**$_2$
0001	&val1	Bucket **01**$_2$
0010	&val2	Bucket **10**$_2$
0011	&val3	Bucket **11**$_2$

Segment **00**$_2$

1000	&val8	Bucket **00**$_2$
1001	&val9	Bucket **01**$_2$
1010	&val10	Bucket **10**$_2$
1011	&val11	Bucket **11**$_2$

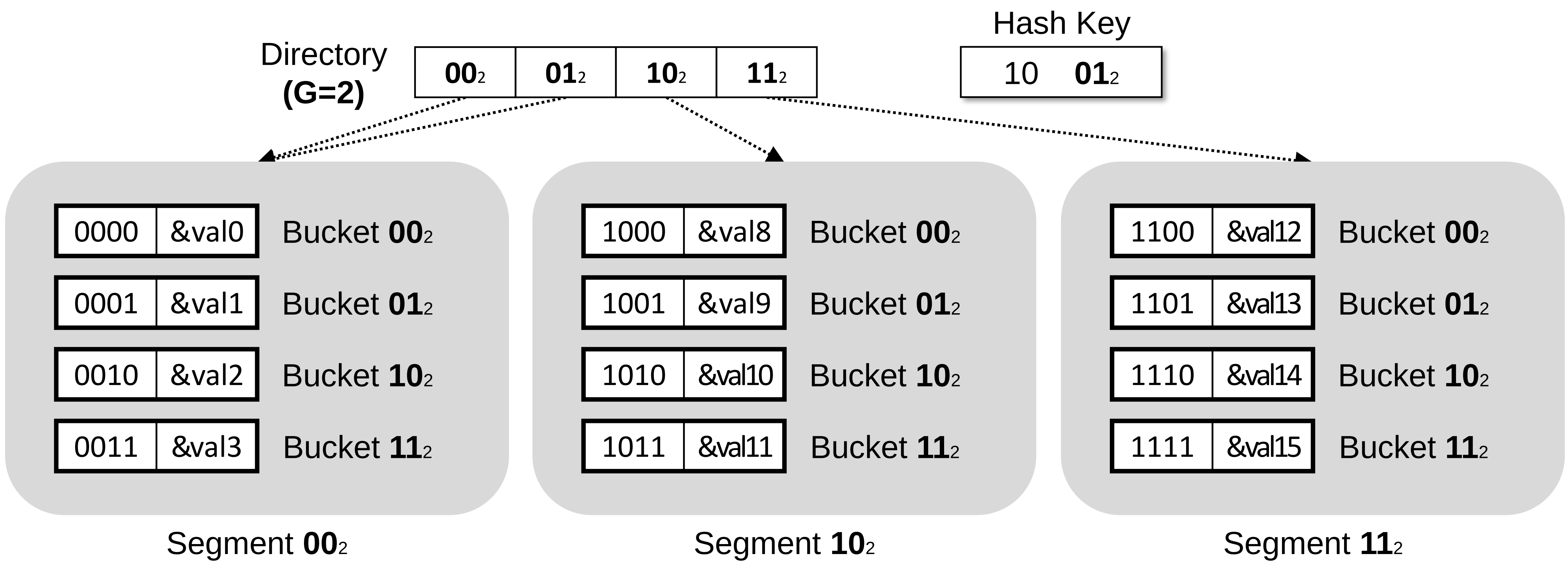
Segment **10**$_2$

1100	&val12	Bucket **00**$_2$
1101	&val13	Bucket **01**$_2$
1110	&val14	Bucket **10**$_2$
1111	&val15	Bucket **11**$_2$

Segment **11**$_2$

Q: With large segments, how can we minimize cacheline accesses?

Minimize Cacheline Accesses in Segment



Directory
(G=2)

$\mathbf{00}_2$	$\mathbf{01}_2$	$\mathbf{10}_2$	$\mathbf{11}_2$

Hash Key

10 $\mathbf{01}_2$

0000	&val0	Bucket $\mathbf{00}_2$
0001	&val1	Bucket $\mathbf{01}_2$
0010	&val2	Bucket $\mathbf{10}_2$
0011	&val3	Bucket $\mathbf{11}_2$

Segment $\mathbf{00}_2$

1000	&val8	Bucket $\mathbf{00}_2$
1001	&val9	Bucket $\mathbf{01}_2$
1010	&val10	Bucket $\mathbf{10}_2$
1011	&val11	Bucket $\mathbf{11}_2$

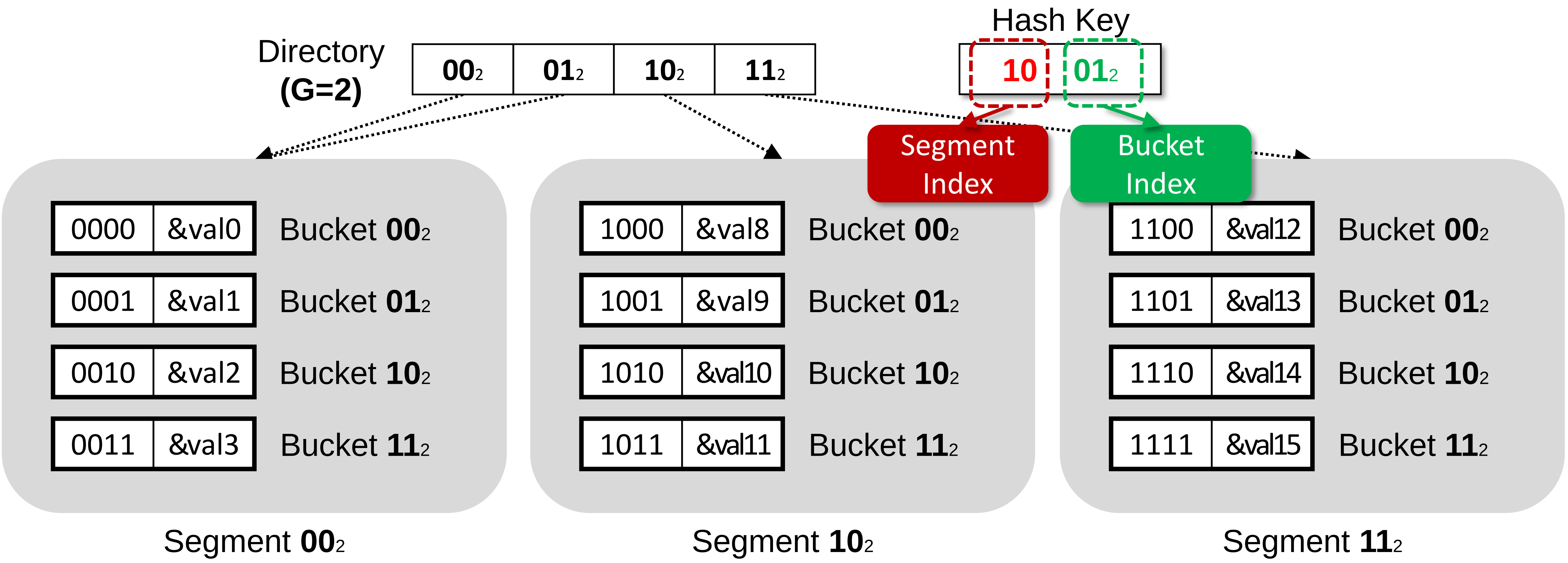
Segment $\mathbf{10}_2$

1100	&val12	Bucket $\mathbf{00}_2$
1101	&val13	Bucket $\mathbf{01}_2$
1110	&val14	Bucket $\mathbf{10}_2$
1111	&val15	Bucket $\mathbf{11}_2$

Segment $\mathbf{11}_2$

Q: With large segments, how can we minimize cacheline accesses?

Minimize Cacheline Accesses in Segment



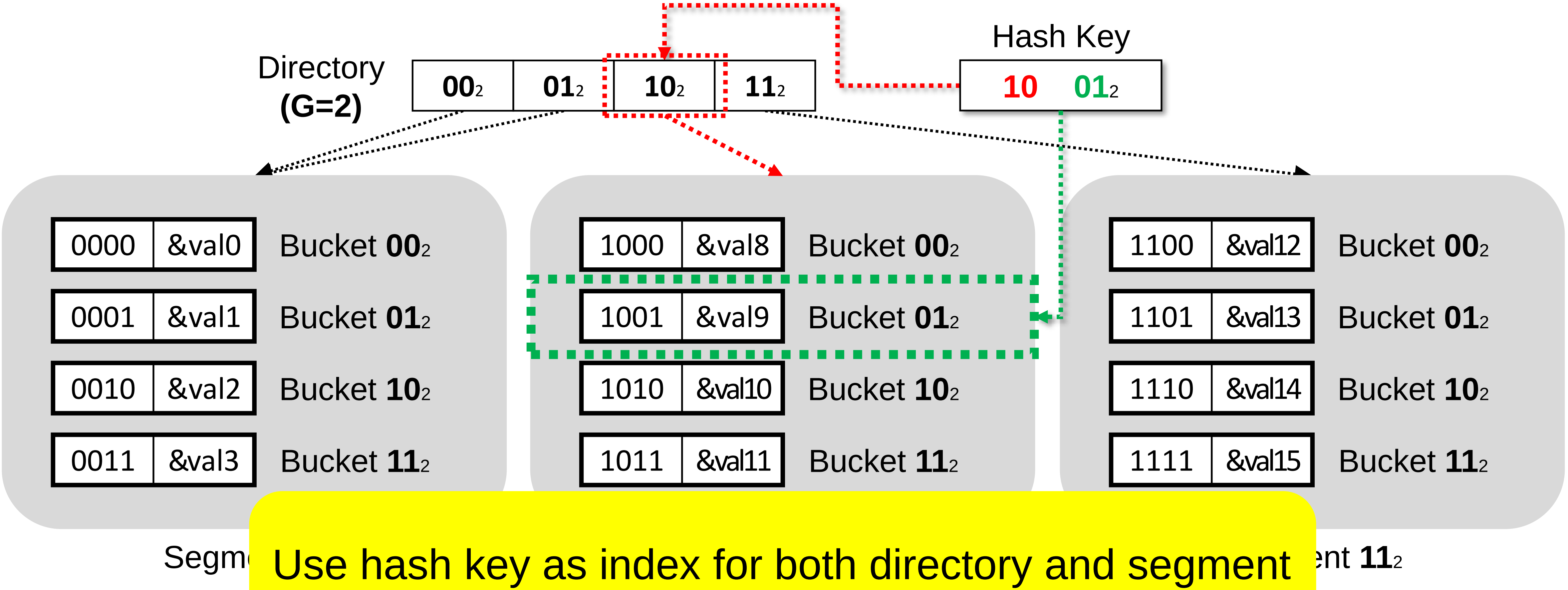
Segment 00_2		
0000	&val0	Bucket 00_2
0001	&val1	Bucket 01_2
0010	&val2	Bucket 10_2
0011	&val3	Bucket 11_2

Segment 10_2		
1000	&val8	Bucket 00_2
1001	&val9	Bucket 01_2
1010	&val10	Bucket 10_2
1011	&val11	Bucket 11_2

Segment 11_2		
1100	&val12	Bucket 00_2
1101	&val13	Bucket 01_2
1110	&val14	Bucket 10_2
1111	&val15	Bucket 11_2

Q: With large segments, how can we minimize cacheline accesses?

Minimize Cacheline Accesses in Segment



0000	&val0	Bucket 00_2
0001	&val1	Bucket 01_2
0010	&val2	Bucket 10_2
0011	&val3	Bucket 11_2

Segm

1000	&val8	Bucket 00_2
1001	&val9	Bucket 01_2
1010	&val10	Bucket 10_2
1011	&val11	Bucket 11_2

1100	&val12	Bucket 00_2
1101	&val13	Bucket 01_2
1110	&val14	Bucket 10_2
1111	&val15	Bucket 11_2

ent 11_2

Use hash key as index for both directory and segment
→ No need to access irrelevant buckets

Contributions

3-Level Structure

→ Introduces an intermediate level, ***Segment***

→ Lookup via only ***two cacheline accesses***

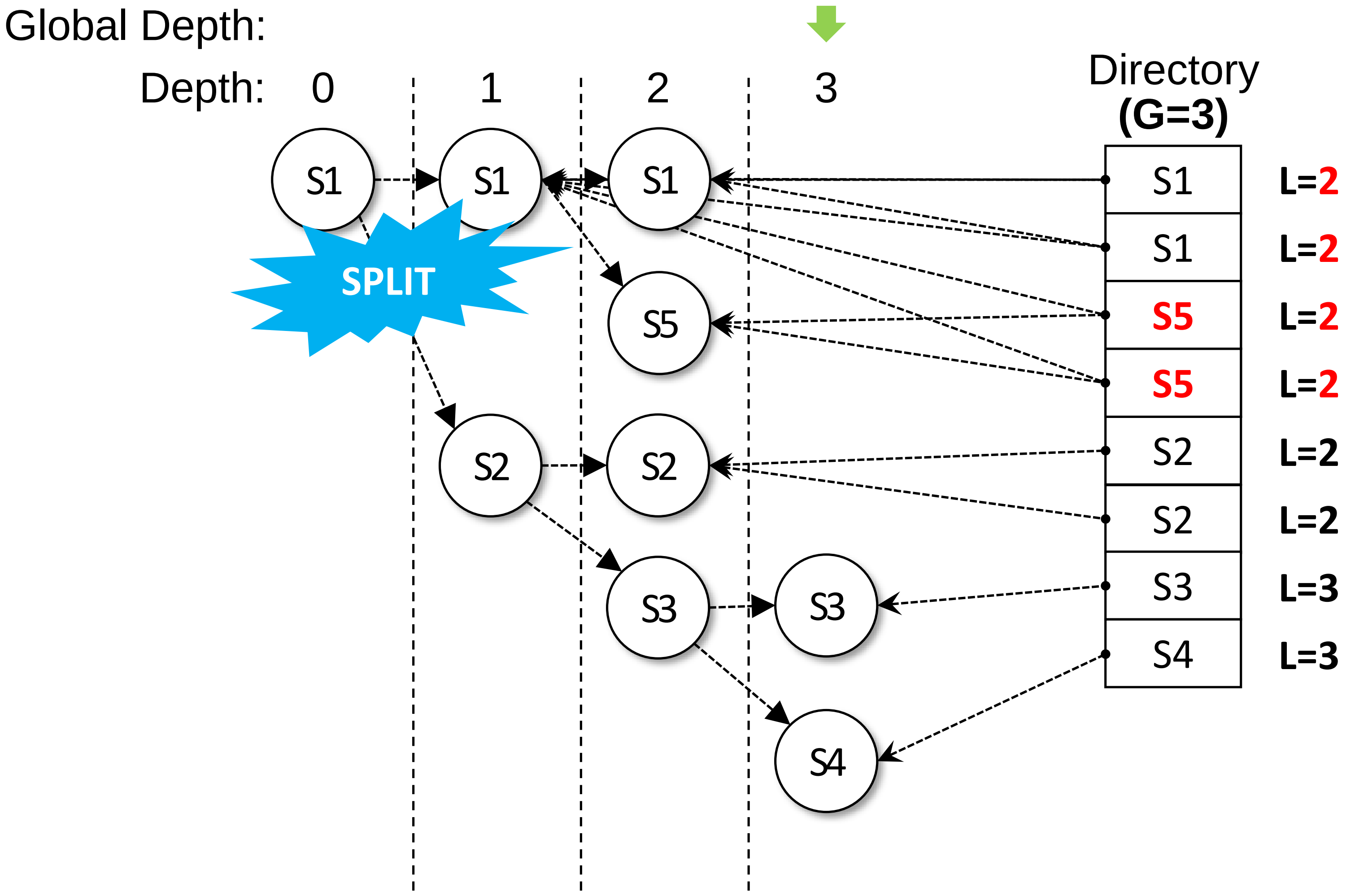
Failure-atomic Directory Updates

→ Introduces ***the split buddy tree*** to **manage split history**

Failure-atomic Segment Split

→ ***Lazy deletion*** scheme to **minimize dirty writes**

Recovery: Split History Buddy Tree in CCEH



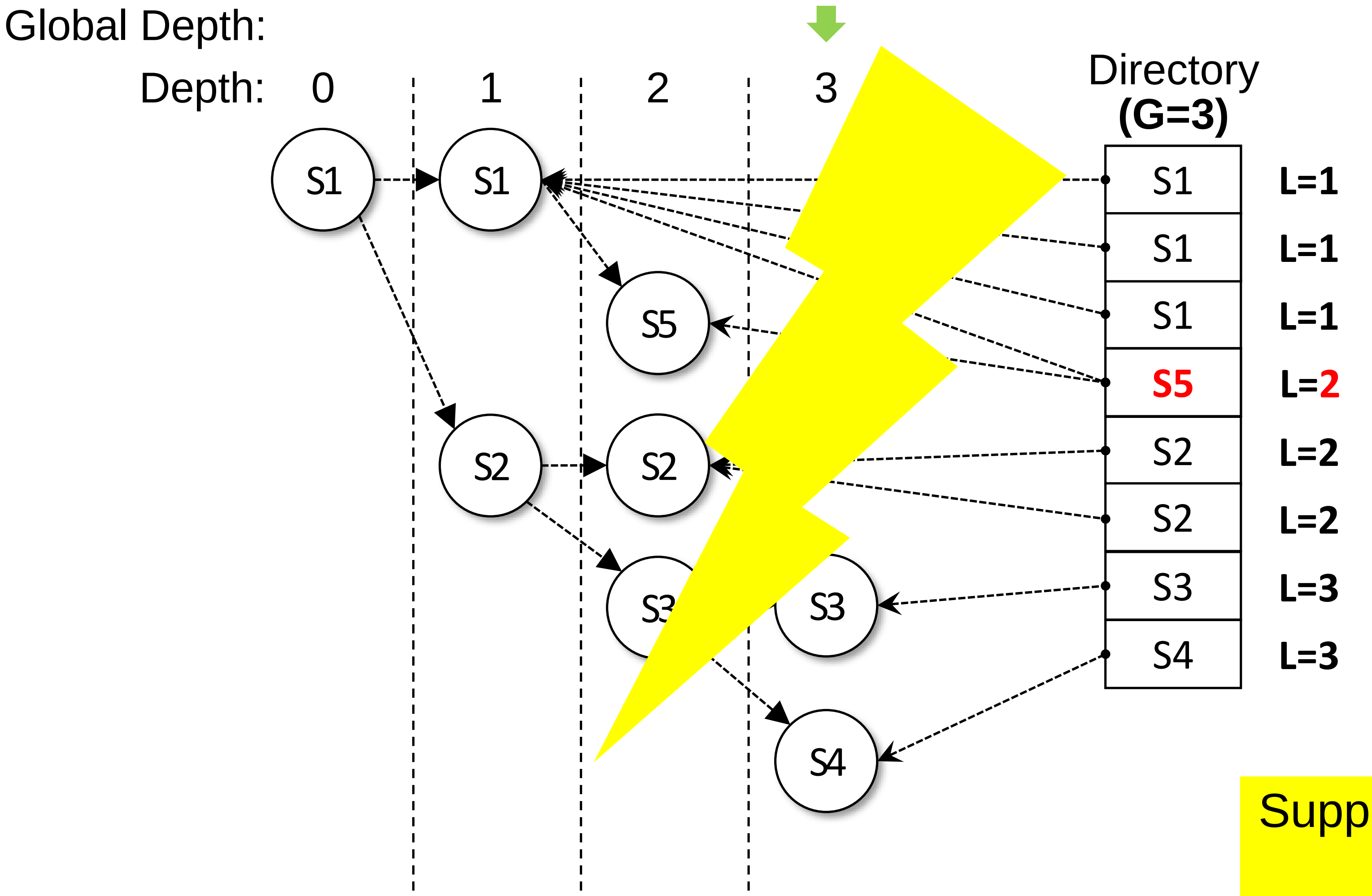
(3) Update local depth of S1

(2) Update a directory entry

(1) Update a directory entry

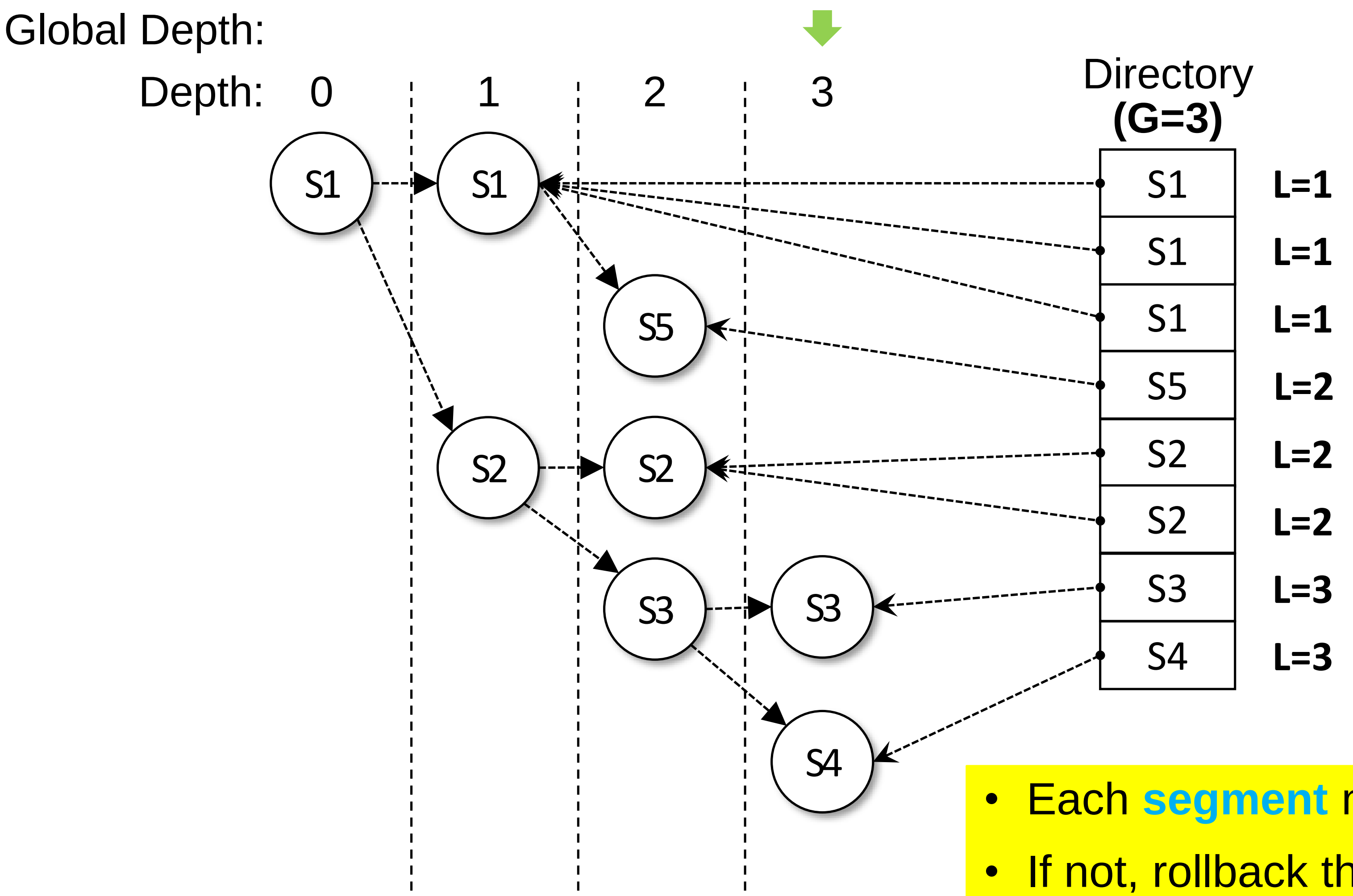
Using **MSB** segment index, split segments are pointed by **adjacent directory** entries

Recovery: Split History Buddy Tree in CCEH



Suppose a system crashes while **S5** splits

Recovery: Split History Buddy Tree in CCEH

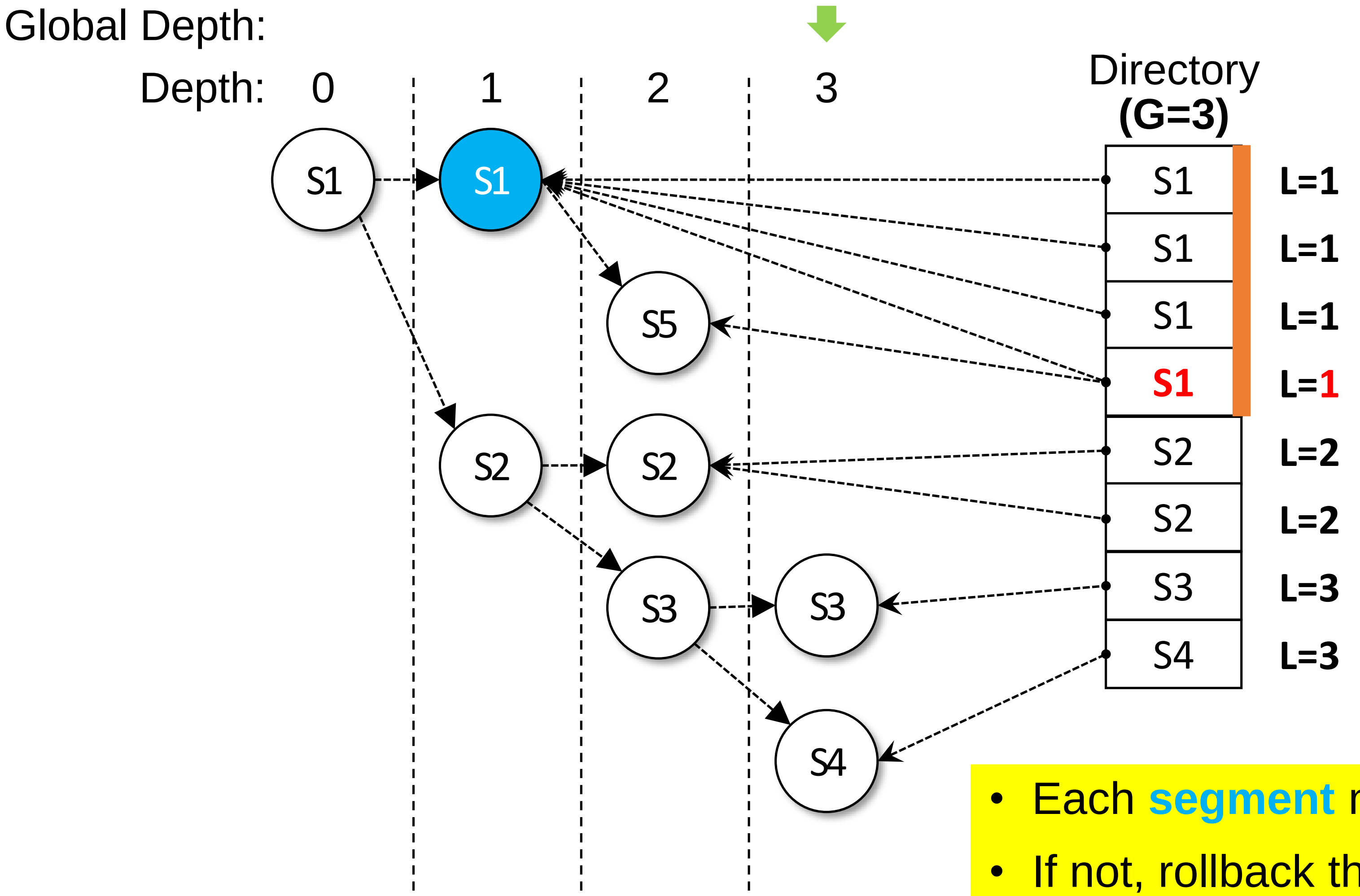


$$\mathit{Stride} = 2^{\mathbf{G-L}}$$

Global Depth **G** = 3
Local Depth **L** = 1
Stride = 4

- Each **segment** must be pointed by $2^{\mathbf{G-L}}$ directory entries
- If not, rollback the split

Recovery: Split History Buddy Tree in CCEH



Directory
(G=3)

S1	**L=1**
S1	**L=1**
S1	**L=1**
S1	**L=1**
S2	**L=2**
S2	**L=2**
S3	**L=3**
S4	**L=3**

$$Stride = 2^{G-L}$$

Global Depth **G** = 3
Local Depth **L** = 1
Stride = 4

- Each **segment** must be pointed by 2^{G-L} directory entries
- If not, rollback the split

Contributions

3-Level Structure

→ Introduces an intermediate level, ***Segment***

→ Lookup via only ***two cacheline accesses***

Failure-atomic Directory Updates

→ Introduces ***the split buddy tree*** to **manage split history**

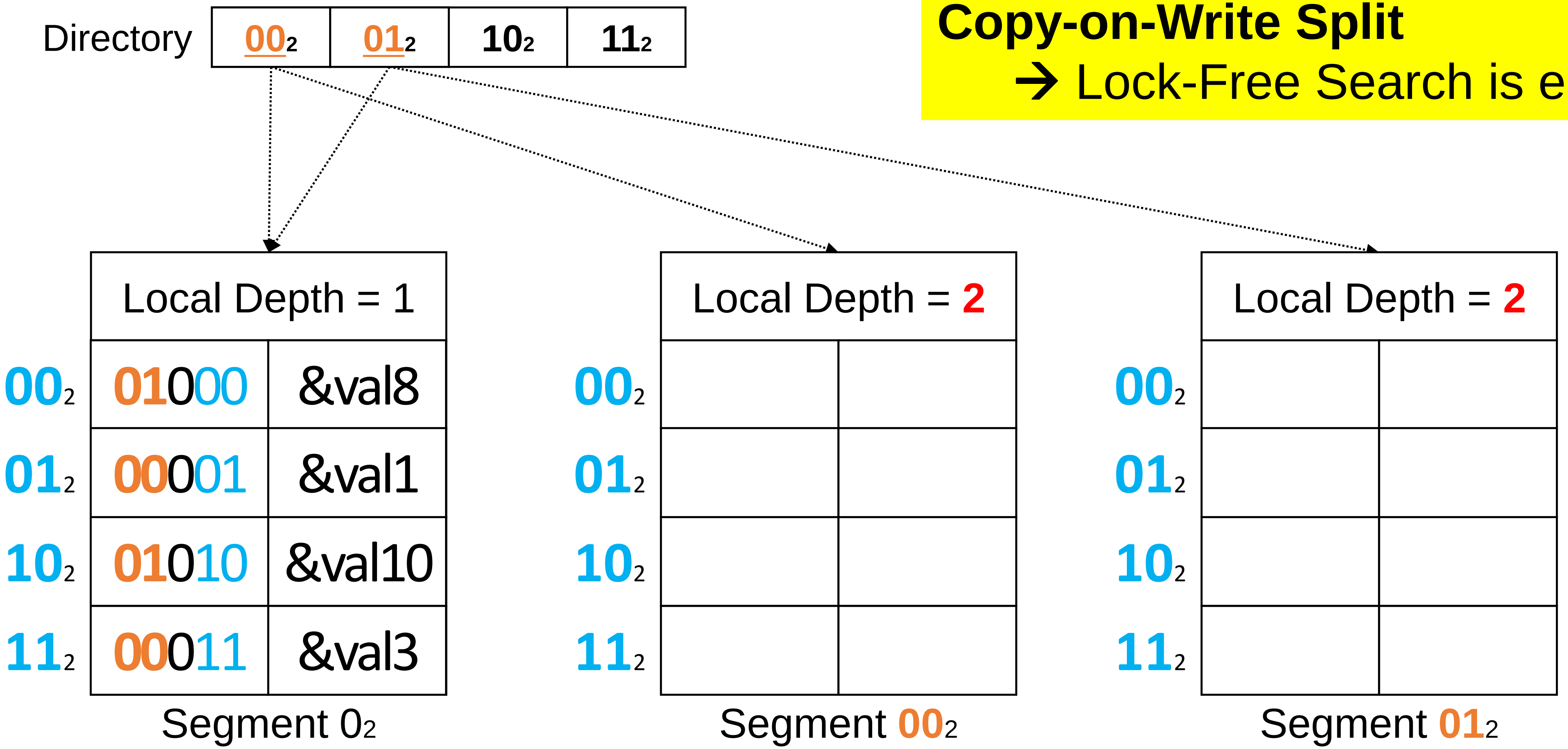
Failure-atomic Segment Split

→ ***Lazy deletion*** scheme to **minimize dirty writes**

Segment Split: Legacy CoW

Copy-on-Write Split

→ Lock-Free Search is enabled



Directory	00_2	01_2	10_2	11_2

	Local Depth = 1	
00_2	01000	&val8
01_2	00001	&val1
10_2	01010	&val10
11_2	00011	&val3

Segment 0_2

	Local Depth = 2	
00_2		
01_2		
10_2		
11_2		

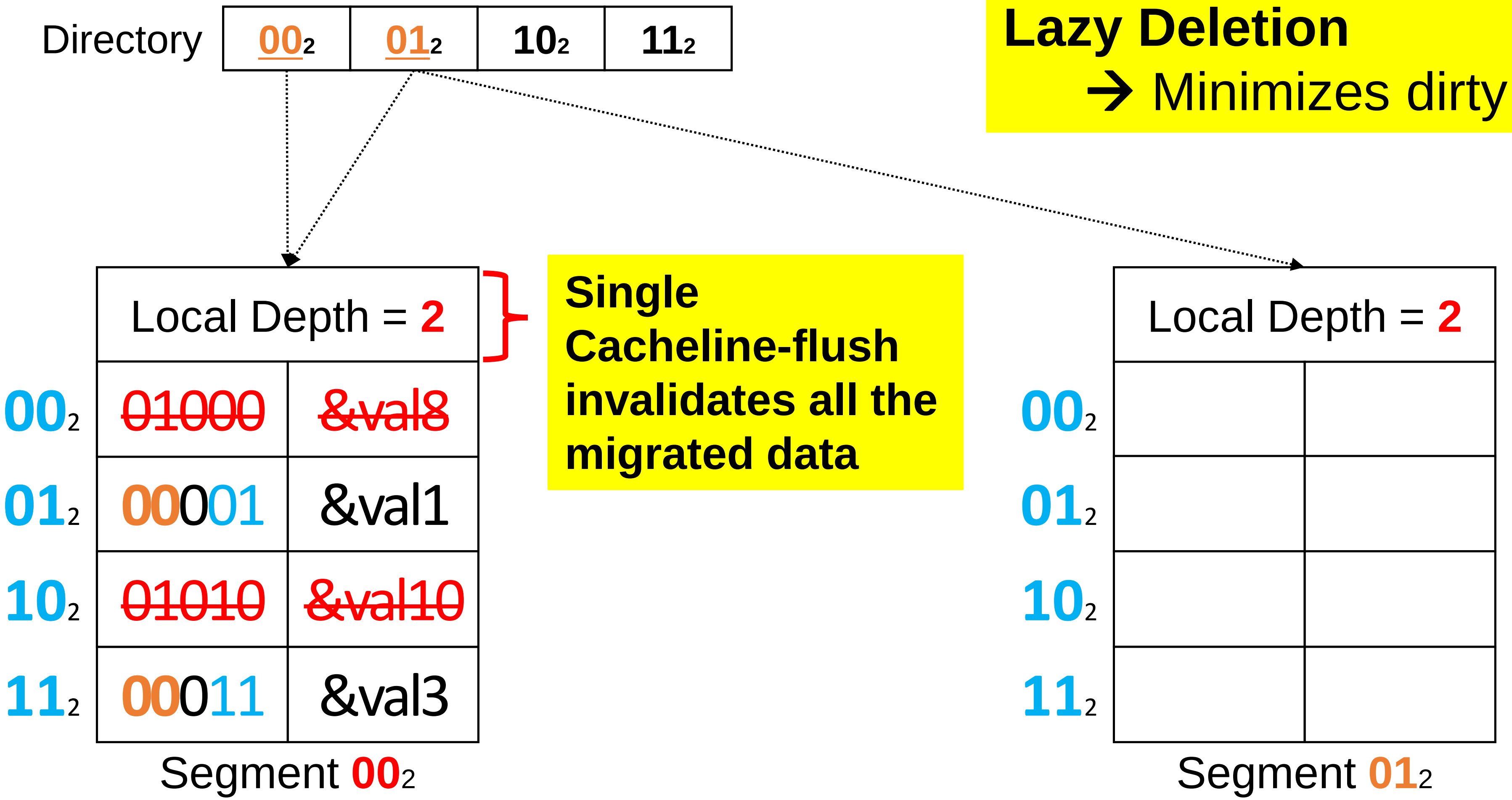
Segment 00_2

	Local Depth = 2	
00_2		
01_2		
10_2		
11_2		

Segment 01_2

Segment Split: Lazy Deletion

Lazy Deletion
→ Minimizes dirty writes



Directory

00_2	01_2	10_2	11_2

Single Cacheline-flush invalidates all the migrated data

	Local Depth = 2	
00_2	~~01000~~	~~&val8~~
01_2	00001	&val1
10_2	~~01010~~	~~&val10~~
11_2	00011	&val3

Segment 00_2

	Local Depth = 2	
00_2		
01_2		
10_2		
11_2		

Segment 01_2

Outline

~~Background~~

- ~~Static Hashing~~
- ~~Extendible Hashing~~
- ~~Persistent Memory~~

~~Cacheline-Conscious Extendible Hashing~~

- ~~Challenges and Contributions~~
- ~~3-Level Structure of CCEH~~
- ~~Failure-atomic Directory Update~~

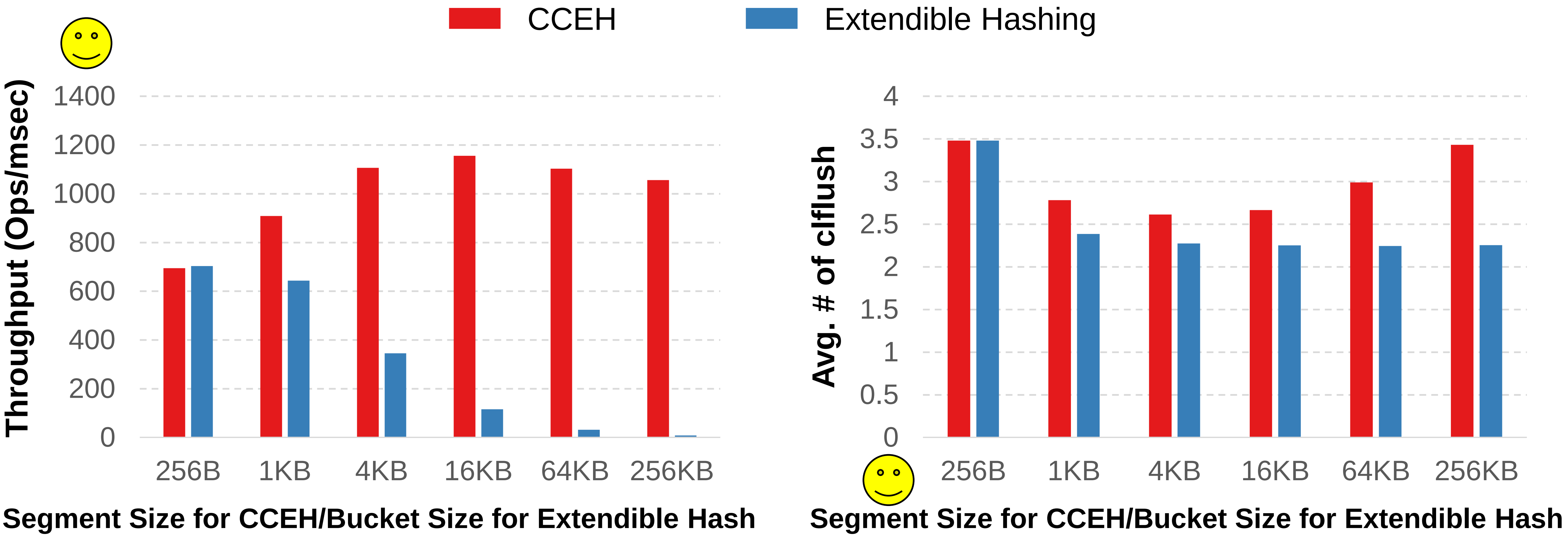
Evaluation

Conclusion

Experimental Setup

CPU	2x Intel Xeon Haswell-Ex E7-4809 v3 → 8 cores, 2.0 GHz → 20MB L3 cache
Memory	64GB of DDR3 DRAM
PM	**Quartz:** A DRAM-based PM latency emulator * To emulate write latency, we inject stall cycle after each ***clflush*** instructions
Workload	160 Million random number dataset

CCEH VS Legacy Extendible Hash

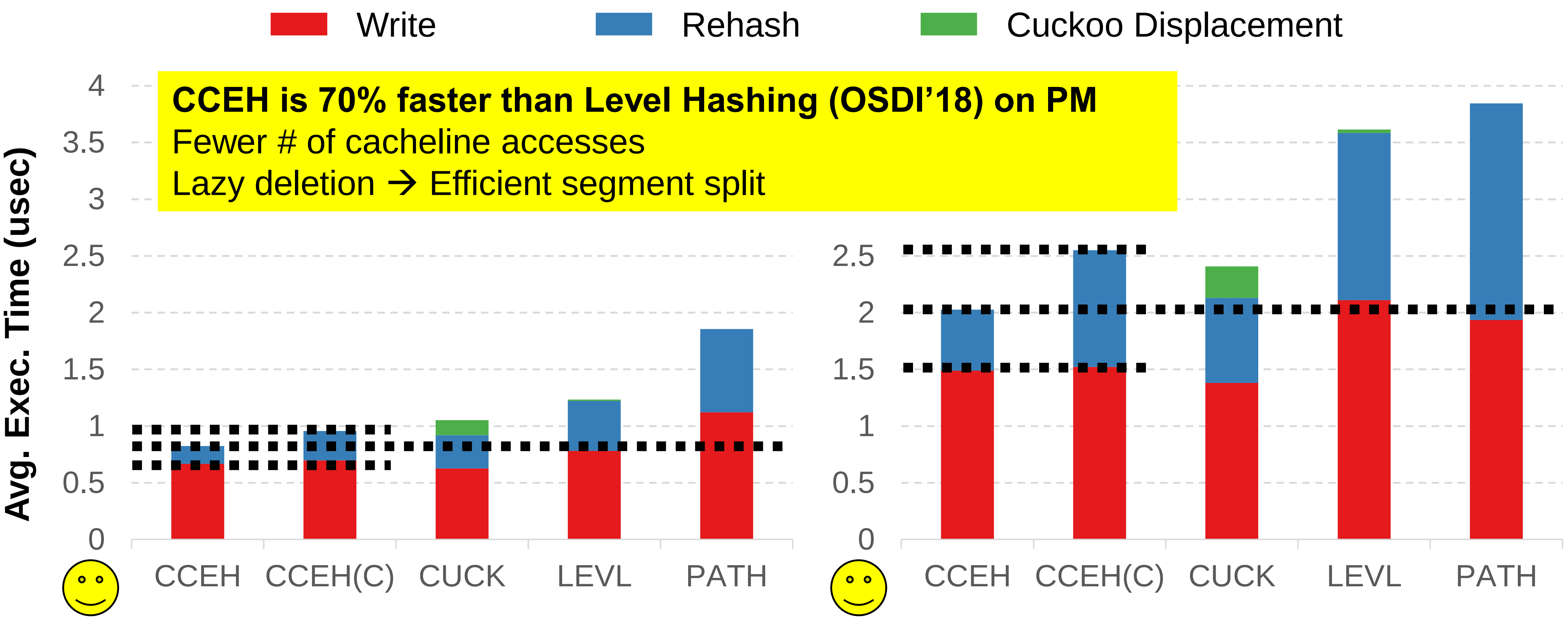


CCEH compared to legacy Extendible Hashing

Cons: Low utilization and more cacheline flushes due to hash collisions

Pros: Constant number of cacheline accesses with varying directory size

Insertion Performance Breakdown



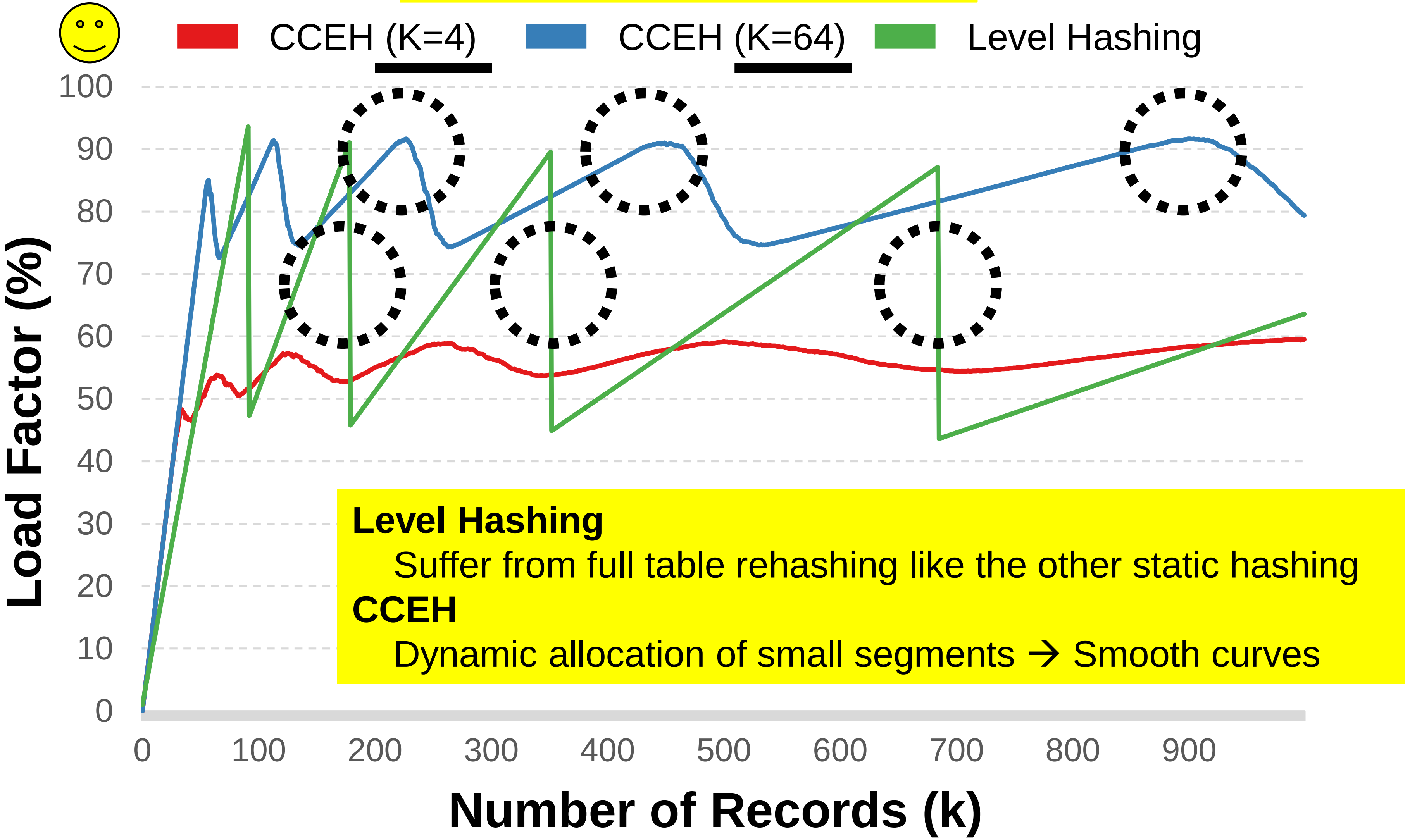
CCEH is 70% faster than Level Hashing (OSDI’18) on PM
Fewer # of cacheline accesses
Lazy deletion → Efficient segment split

R:120ns/W:120ns
(DRAM)

R:240ns/W:700ns

Load Factor

CCEH (Optimization**)**
K = Linear probing distance in Segment



Level Hashing

Suffer from full table rehashing like the other static hashing

CCEH

Dynamic allocation of small segments → Smooth curves

Conclusion

Cacheline-Conscious Extendible Hashing (CCEH)

- 3-Level Structure
  - Introduced an intermediate level, Segment
  - Constant Lookup: Only two cacheline accesses → Write-Optimal

- Failure-Atomic Write-Optimal Lazy Deletion → Minimize I/O
- Failure-Atomic Directory Updates → Log-less directory update

Disk-based hashing needs to be modified for PM to make effective use of cachelines.

Source Codes: http://github.com/DICL/CCEH

Question?

рахмат
danke
謝謝
ngiyabonga
Баярлалаа
spas
teşekkür ederim
mersi
kia ora
barka
welalin
tack
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
hvala
köszönöm
gràçie
bayarlalaa
bedankt
thank you
aciu
dankon
akun
gracias
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
enkosi
djiere dieuf
tau
mochchakkeram
дякую
mamnun
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
děkuji
obrigado
sagolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
terima kasih
kam sah hamnida
rahmat
dhanyavadagalu
shukriya
mercé
мерси
diolch
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci