筑波大学
University of Tsukuba

Stream-BSGS

Memory-Efficient Tree-BSGS with Optimized Relinearization Scheduling for FHE Polynomial Evaluation

Zhongqi Wang [1] Shintaro Narisada [2] Takashi Nishide [1]

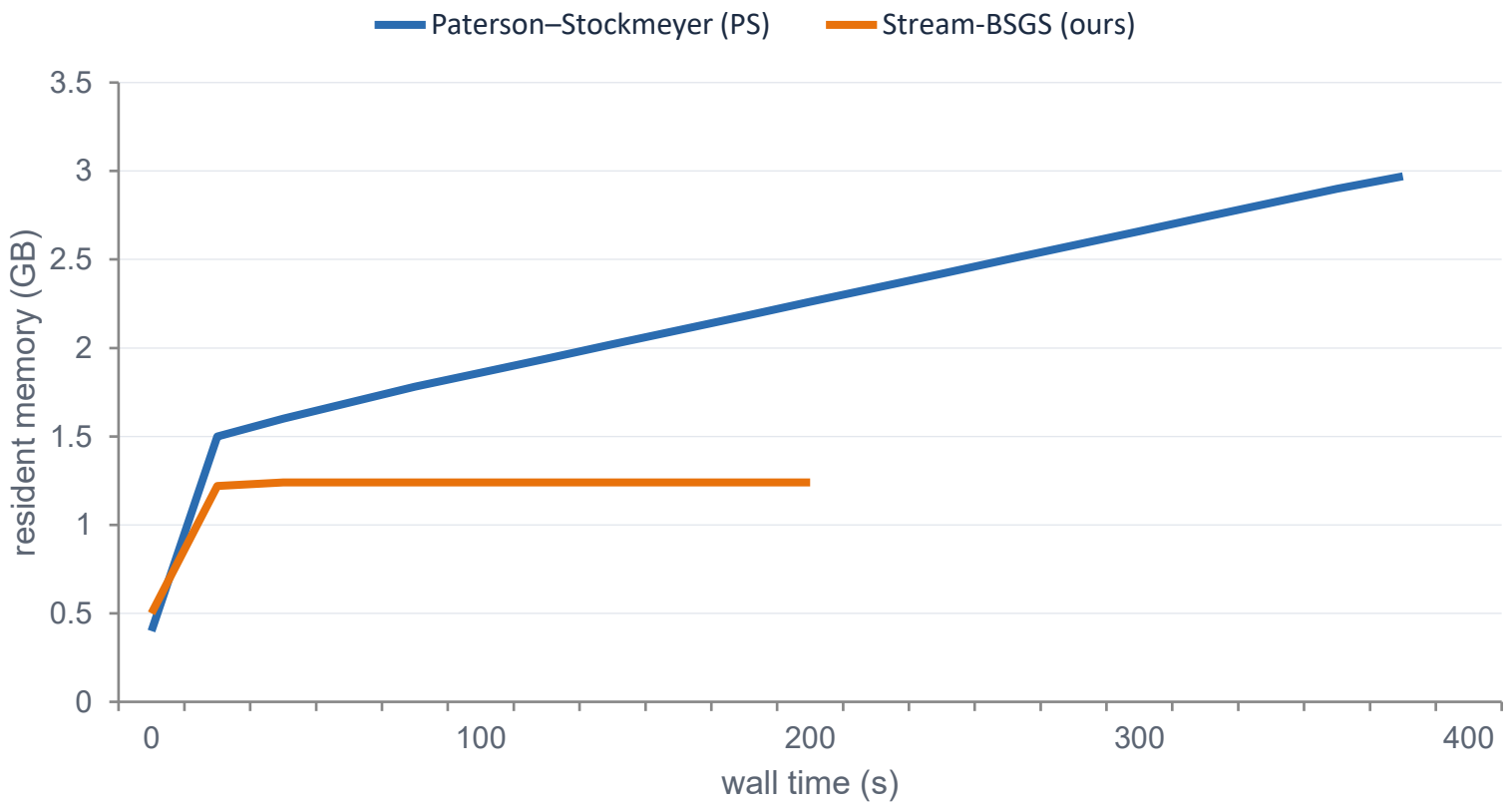
[1] University of Tsukuba [2] KDDI Research, Inc.

SAC 2026 · University of Ottawa · August 28, 2026

Recorded talk — I will join live on Zoom for Q&A

THE RESULT

One polynomial evaluation, two ways — half the time, a fraction of the memory



CKKS, degree 2^{15} (N = 32768, depth 17) — endpoints from Table 5 of the paper

1.9×

faster

2.4×

less peak memory

…computing exactly the same function.

"clean up" = relinearization — FHE's mandatory tidy-up after each multiplication (slide 4).

Only two things changed: WHEN we clean up after multiplication — and in WHAT ORDER we run.

THE MAP

One computation — two knobs left to turn

Where high-degree polynomial evaluation shows up:

Exact comparison — BFV / BGV

interpolation over the plaintext space
→ degree ≈ 2^{16} for 16-bit values

High-precision approx. — CKKS

sign, ReLU, …
→ degrees in the thousands

CKKS bootstrapping

same BSGS machinery,
at moderate degrees (~10^2)

The FHE multiplications themselves

fixed by the polynomial — same for every method

Knob ① WHEN to clean up

relinearization scheduling → PART 1

Knob ② in WHAT ORDER to execute

streaming execution → PART 2

Spoiler: knob ② is what makes knob ① pay off.

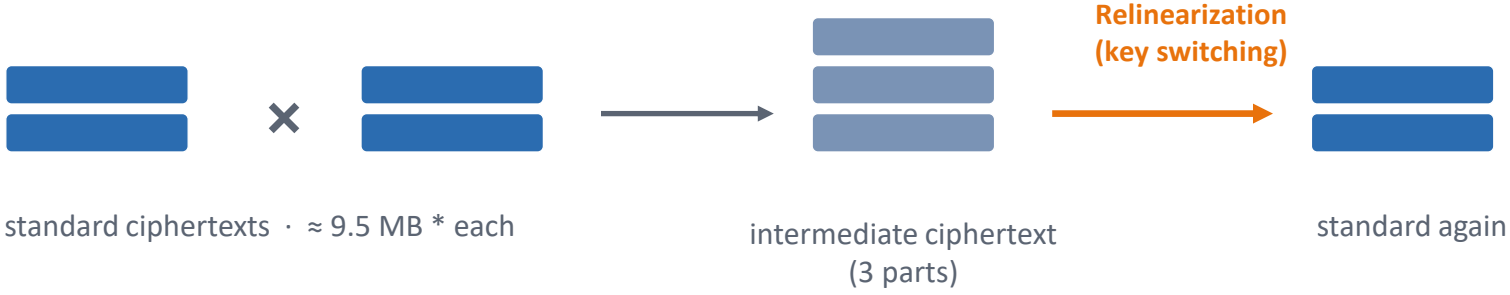
Part 1 finds the best schedule — Part 2 makes it show up end to end.

Multiplication makes ciphertexts fatter — relinearization slims them, expensively

① **a ciphertext = 2 big ring elements — ≈ 9.5 MB (CKKS/BGV) · ≈ 5 MB (BFV) ***

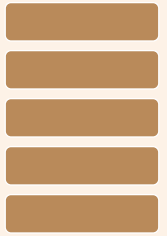
② **multiply → 3 parts, not 2 — an intermediate, fatter ciphertext**

③ **relinearization squeezes it back — and costs ≥ the multiplication itself**



cost ≥ the multiplication itself

Skip the cleanup?



much fatter ciphertext

Allowed — 4, 5, … K parts.
But meanwhile, heavier keys
are needed to clean up later.

* N = 32768, depth 17 → ≈ 9.5 MB per ciphertext (CKKS/BGV), ≈ 5 MB (BFV) — i.e. 4.75 / 2.5 MB per ring element · hybrid key switching: dnum digit products + 2 base conversions

Every multiplication leaves a mess — cleaning it is FHE's costliest routine step.

BSGS: split into ≈ √d chunks, combine up a binary tree

Parameters: d = polynomial degree · split into s chunks of b coefficients each → b × s ≈ d, b ≈ s ≈ √d (at $d = 2^{16}$: b = s = 256)

1 Split

coefficients → s chunks of b each (b ≈ s ≈ √d)

2 Baby powers

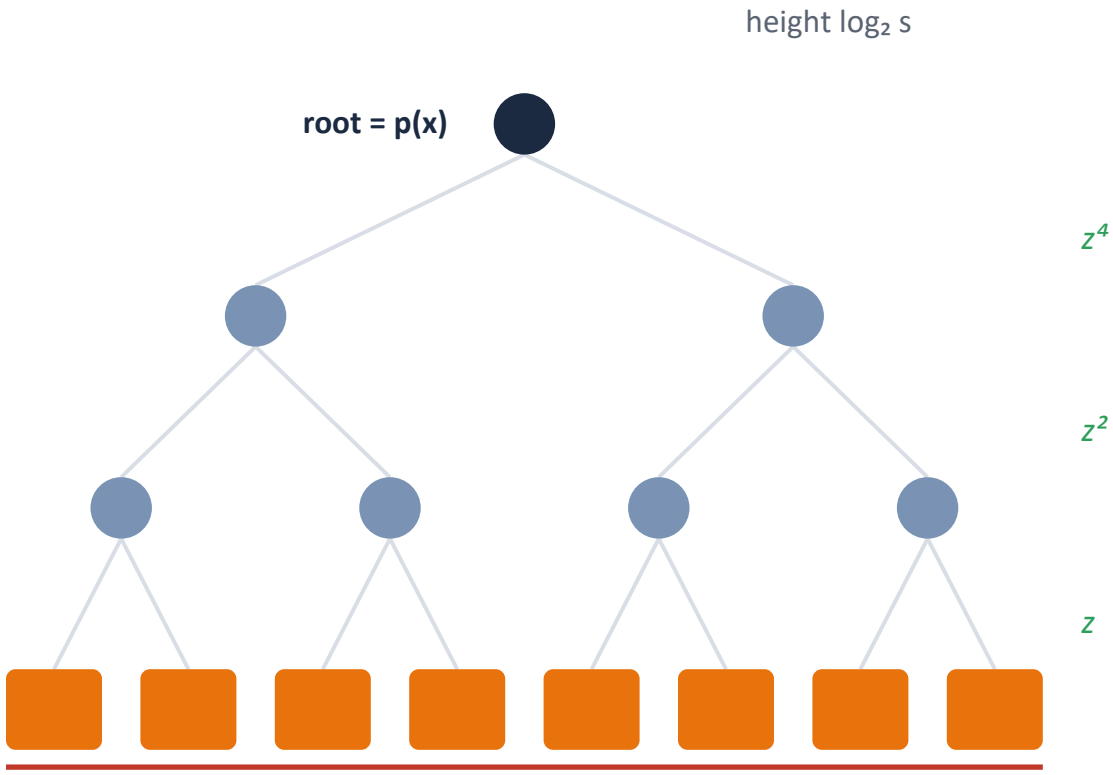
x, x^2, …, x^b — expensive ct×ct work

3 Leaves

each chunk → one leaf, via cheap ct×plaintext ops

4 Combine

merge with giant powers z^(2^ℓ) · classical PS: a chain, s−1 extra ct×ct mults



s = 8 leaves — each ■ = one ciphertext (≈ 9.5 MB) — **all resident at once**

The catch in every existing BSGS: all s leaves must sit in memory at once.

Tree-BSGS is the state of the art — but every leaf is a full ciphertext, and ALL of them stay live at once.

We score algorithms on four axes — adding peak memory

Parameters: d = degree · b = chunk size · s = number of chunks (= leaves) · $b \approx s \approx \sqrt{d}$ (at $d = 2^{16}$: $b = s = 256$)

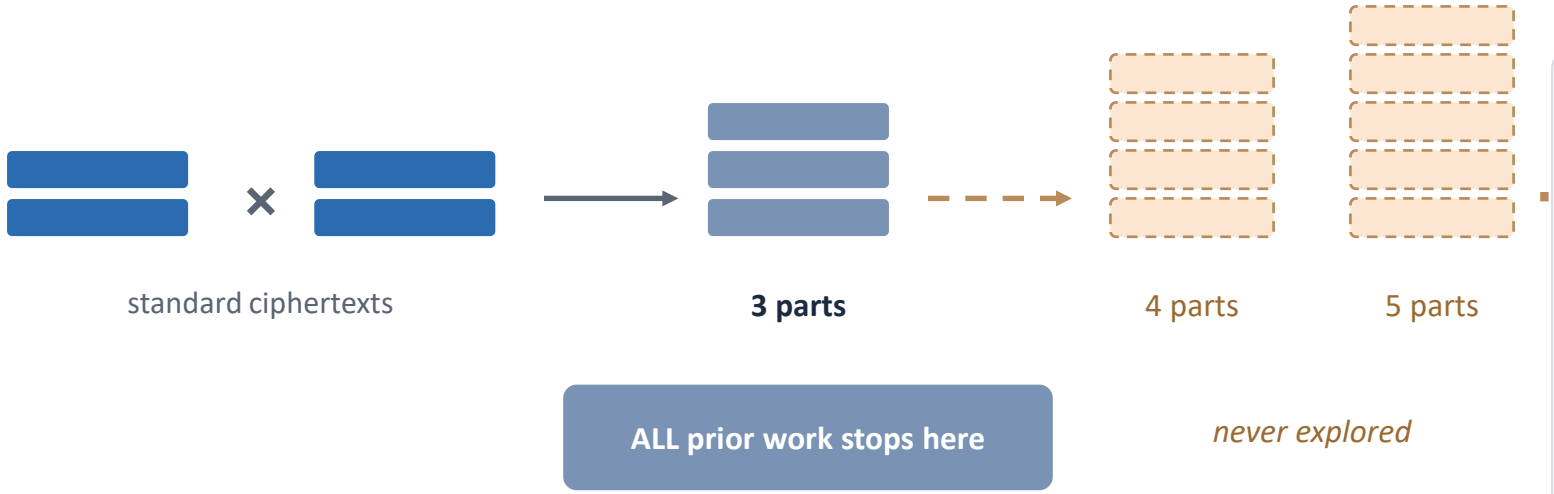
	ct×ct mults	mult. depth	relin count	peak working set
PS	$b + 2s - 3$	$\lceil \log_2(d-\sqrt{d}) \rceil + 1$	$b + 2s - 3$	**$b + 2s - 1$ = 767 @ $d = 2^{16}$**
Tree-BSGS	$b + s - 2$	$\log_2 d$	$b + s - 2$	**$b + s + \log_2 s$ = 520**

Peak working set = maximum number of ciphertexts alive at the same time.

Tree-BSGS wins on the three classical axes. But both methods keep a term of size **$s \approx \sqrt{d}$** in memory — tree-BSGS stores **all the leaves** before it starts combining. Does that term have to be there?

Each live ciphertext is 5–9.5 MB — this fourth column is literally your memory bill.

Everyone defers cleanup by exactly one level. Why stop there?



defer ONE level: let it reach 3 parts — then relinearize back to 2

All prior implementations:

Han & Ki (CT-RSA '20)	1 level
Bossuat et al. (EuroCrypt '21)	1 level
Lee et al. (EuroCrypt '22)	1 level
Wang et al. (CANDAR '25)	1 level

What if we defer deeper?

Everybody postpones the cleanup by one step — nobody asked what happens if you postpone harder.

One cleanup step costs more than 2× one extra unit of mess

one extra product part (Δmul)

= 2 ring multiplications

"one unit of mess"

one key-switch step (Δrelin)

= fixed bundle: digit products + 2 base conversions

"one unit of cleanup"

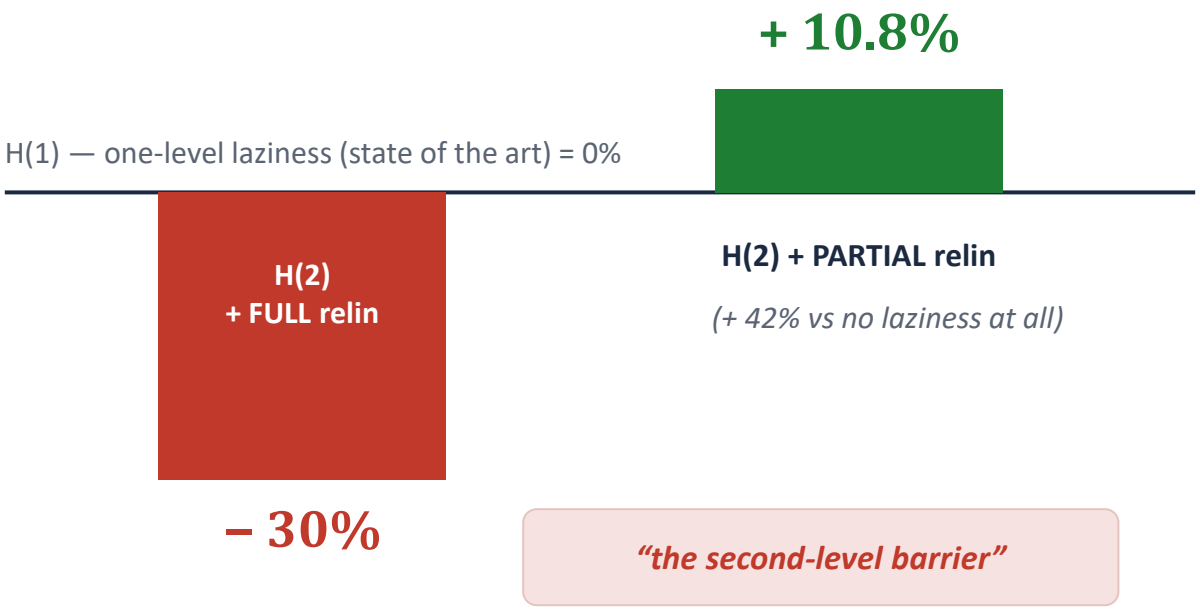
$\alpha = \Delta\text{relin} / \Delta\text{mul} > 2$ for every standard parameter set · $> 10/3$ at ours

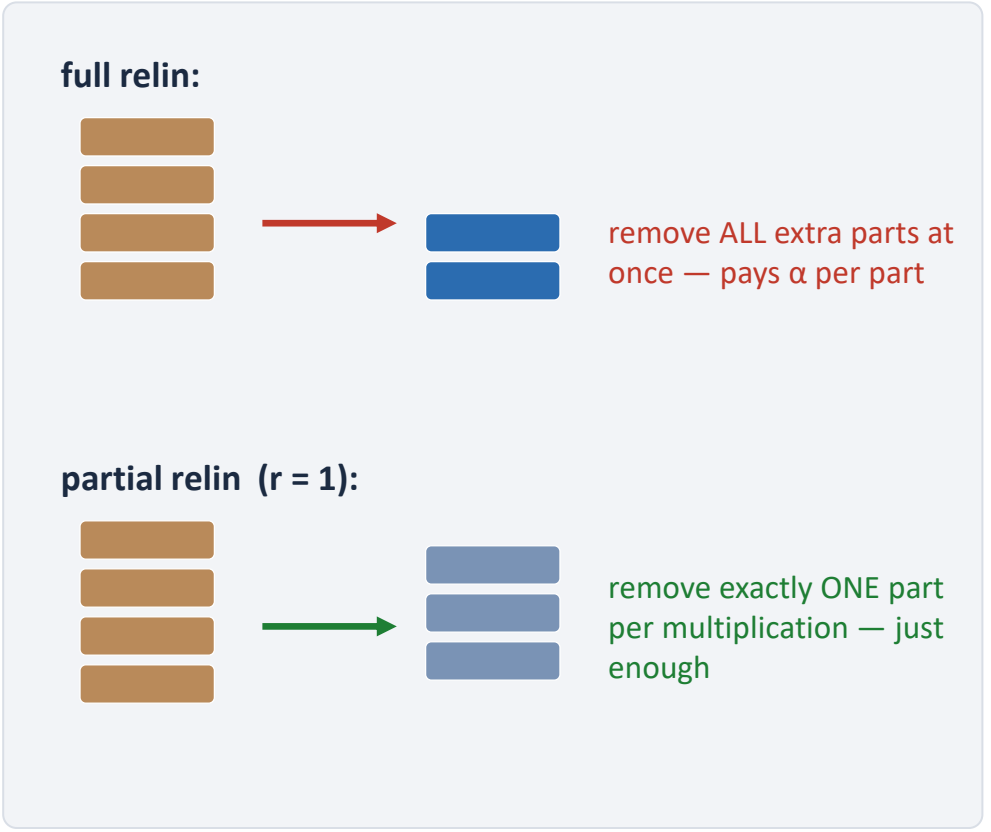
both marginal costs independent of the part count K (Prop. 8) — BFV / BGV / CKKS, hybrid key switching

⚠ **Necessary, NOT sufficient:** $\alpha > 2$ makes deferring promising — but defer deeper naively (FULL cleanup) and you get SLOWER.

Cleanup costs ≥ 2× the mess — necessary for laziness to win, but not yet sufficient.

Full cleanup makes deeper deferral backfire — partial cleanup flips it





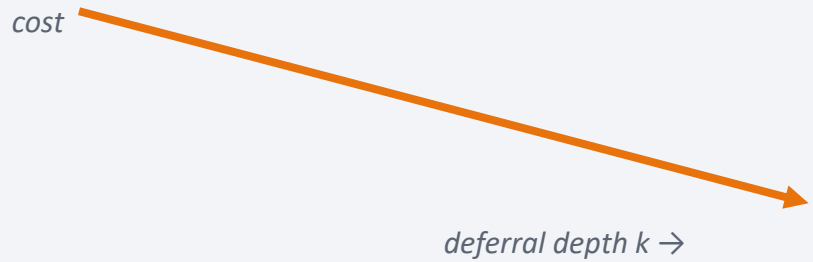
CKKS bootstrapping parameters, tree size ≥ 256 (paper Table 3)

Never clean more than you must — the wall was the cleaning style, not the laziness.

Optimal policy: relinearize as little as possible, as late as the noise allows

Deeper is cheaper — always

Under partial relin, total cost strictly decreases with the deferral depth k (Theorem 12).



BFV / BGV : go deeper until decryption fails → **k* = 6**

CKKS : go deeper until precision drops → **k* = 2**

depth k	output precision (CKKS)	OK?
0 · 1 · 2	31.8 bits	✓
3	26.9 bits (− 5 bits)	✗
4	18.0 bits (− 14 bits)	✗

↑ *shown: the CKKS case*

k* (“k_noise”) is found by O(log d) cheap trial runs — and proved optimal within level-wise schedules (Thm 18)

That single sentence is Part 1 — worth 42% in the tree stage. But…

THE CATCH

End-to-end, the scheduling gain vanishes — memory eats it

Where the time actually goes:

← the phase our schedule accelerates

scheduling alone, end-to-end:

0.97× – 1.02× (≈ nothing)

peak memory (BFV/BGV):

up to 0.90× — it gets WORSE

deeper laziness → fatter live ciphertexts + larger keys → **memory pressure on the dominant phase**

Ablation, degree 2^{15} (paper Table 6)

We won the battle in the tree — and lost the war in memory. Fix the memory → Part 2.

PART 2 — IN WHAT ORDER TO RUN

The √d-sized memory bill is an artifact of phase separation

Phase A — generate ALL s leaves

each ■ = one ciphertext — all s ≈ √d of them alive at once (≈ 9.5 MB each)

Phase B — reduce the tree

…consumes the leaves two at a time, in order.

Each leaf could be consumed the moment it is born.

So: fuse the phases.

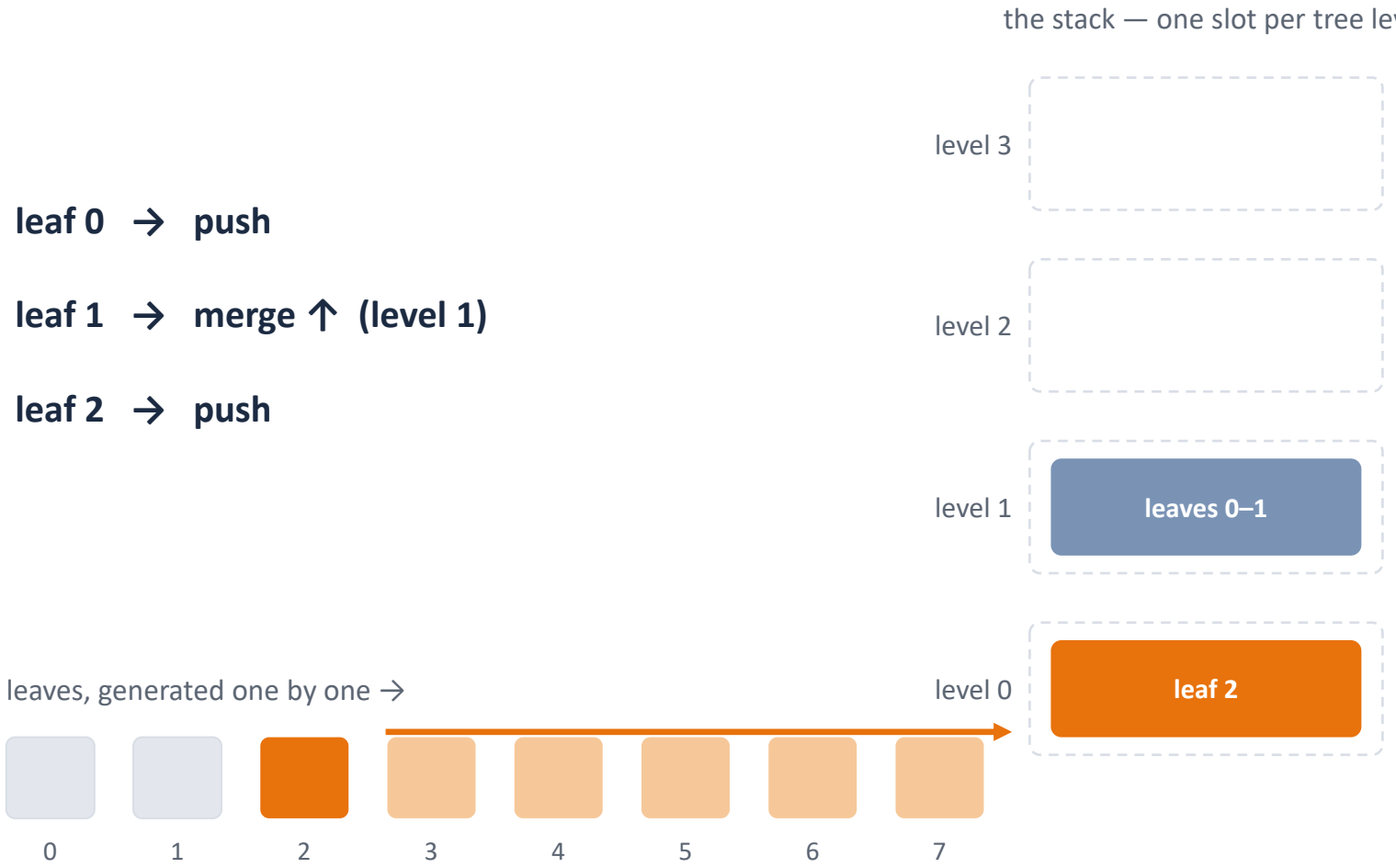
Nothing in the math needs all √d leaves alive together.

Stream-BSGS: push, merge, forget — one leaf at a time

leaf 0 → push

leaf 1 → merge ↑ (level 1)

leaf 2 → push



binary counter

$$j = 3 = 011_2$$

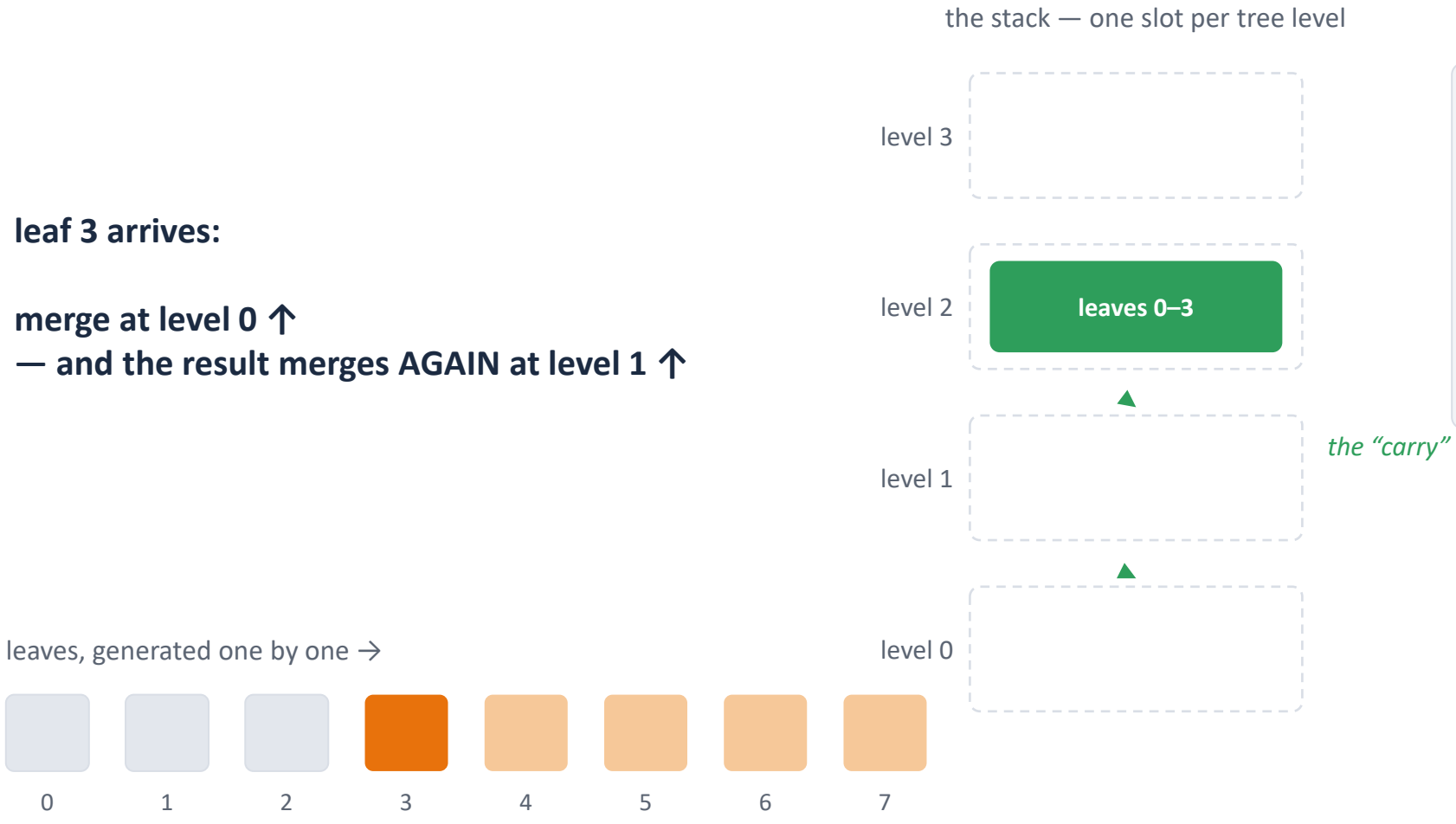
stack size = # of 1-bits = **2**

Fold each leaf the moment it is born — then forget it.

The carry ripples upward — exactly like binary +1

leaf 3 arrives:

merge at level 0 ↑
— and the result merges AGAIN at level 1 ↑



binary counter

$j = 4 = 100_2$

stack size = # of 1-bits = **1**

After j leaves: one stack node per 1-bit of j → stack ≤ log s.

Stack never exceeds log s — and inherits the cleanup rule for free

Inside every merge:

the same partial-relin rule from Part 1 — literally the same code path. Stream-BSGS inherits the optimal schedule for free.

= depth-first (post-order) traversal of the same tree · a classic streaming fold

the stack — one slot per tree level

level 3 — ROOT = p(x)

level 2

level 1

level 0

binary counter

$$j = 8 = 1000_2$$

stack size = # of 1-bits = **1**

leaves, generated one by one →

0 1 2 3 4 5 6 7

Same schedule, inherited for free — while the stack stays ≤ log s.

PART 2 — WHAT IT BUYS

Peak drops from 520 to 274 live ciphertexts — arithmetic bit-identical

	peak working set	live ciphertexts @ degree 2^{16}
PS	$b + 2s - 1$	767
Tree-BSGS	$b + s + \log_2 s$	520
Stream-BSGS	**$b + 2\log_2 s + 2$**	**274**
Stream-BSGS (b-halved)	**$b/2 + 2\log_2 2s + 2$**	**148**

$b = s = 256$ · ≈ 9.5 MB per live ciphertext (CKKS/BGV)

Exact reordering — proved

same multiplications · same depth
same noise · same output
→ drop-in for BFV, BGV, CKKS

Bonus — the b-dial: peak now depends on b, not s. Halve b → memory ↓ another ~1.5×, time + a few %.
PS / tree-BSGS cannot do this — their peak is dominated by s.

Same math, a fraction of the memory — plus a memory dial you can turn.

THE PAYOFF — END TO END

1.66–1.91× faster than PS · 1.65–2.81× less memory — all three schemes

OpenFHE v1.2.3 · N = 32768 · depth 17 · degrees 2^{14}, 2^{15}, 2^{16} · every run verified against a plaintext computation

scheme	speedup vs PS	peak-memory reduction
CKKS	**1.89× – 1.91×**	1.65× – 2.40×
BFV	**1.66× – 1.74×**	1.77× – 2.81×
BGV	**1.79× – 1.84×**	1.71× – 2.48×

b-halved variant: memory ↓ up to **3.7×** — still ≥ 1.49× faster than PS

Remember slide 2?

The flat orange line
is this table.

code + benchmarks:
github.com/WayneOnEarth/
Stream-BSGS

The flat orange line from slide one — now you know why it is flat.

THE PAYOFF — ABLATION

Identical arithmetic, streaming order: 1.4–1.9× faster

configuration (degree 2^{15})	end-to-end time	peak memory
Tree-BSGS	baseline	baseline
+ optimal scheduling (Part 1)	0.97× – 1.02×	0.90× – 1.15× (can worsen)
Stream-BSGS (Part 1 + 2)	**1.38× – 1.94×**	**1.44× – 2.13× ↓**

rows 2 and 3 execute the identical ciphertext-level operations — only the execution order differs

In FHE, every object weighs megabytes. How many you keep alive is **not an implementation detail.**

Peak working set belongs in the algorithm's scorecard — next to the multiplication count.

Three things to remember

1 **Relinearize as little as possible, as late as the noise allows**

provably optimal — and it explains the one-level tradition

2 **Stream the tree — fold each leaf as it is born**

same math, a fraction of the memory; this is what turns scheduling into end-to-end speedup

3 **Count your live ciphertexts**

peak working set as a first-class dimension for comparing FHE algorithms

Future work: analytic model for k_noise · per-node schedules · full bootstrapping pipeline

Thank you — joining you live on Zoom for questions now.