Developers' Meeting

BERLIN 2025

mobileye™

Planning Tile & Fuse Transform in MLIR

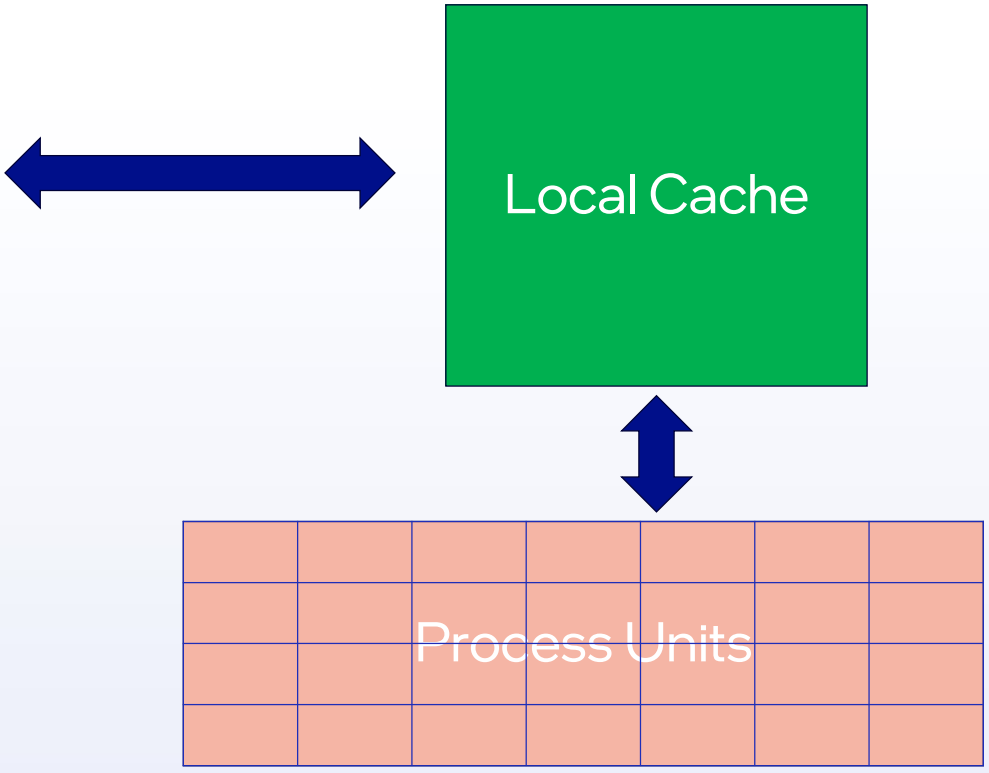
April 8, 2025

Aviad Cohen, Algorithm engineer

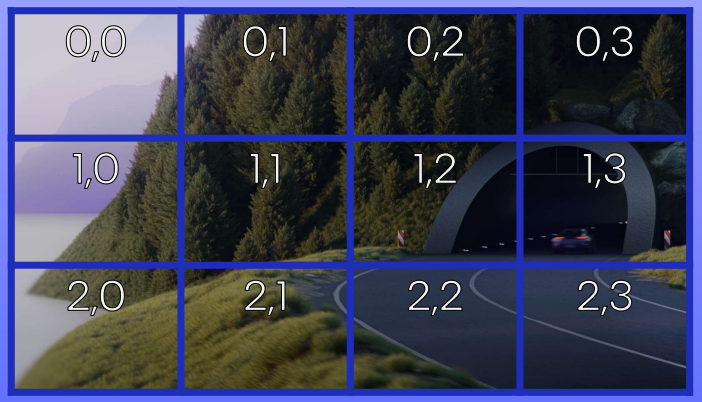
1 Picture, 1000 words

What is tiling all about?

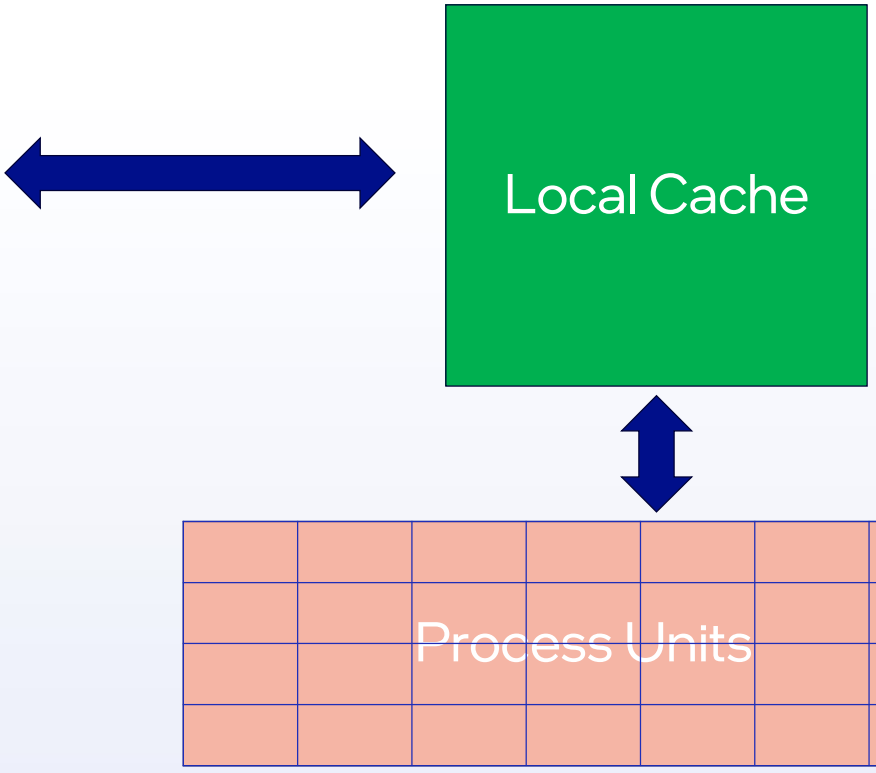
Main Memory



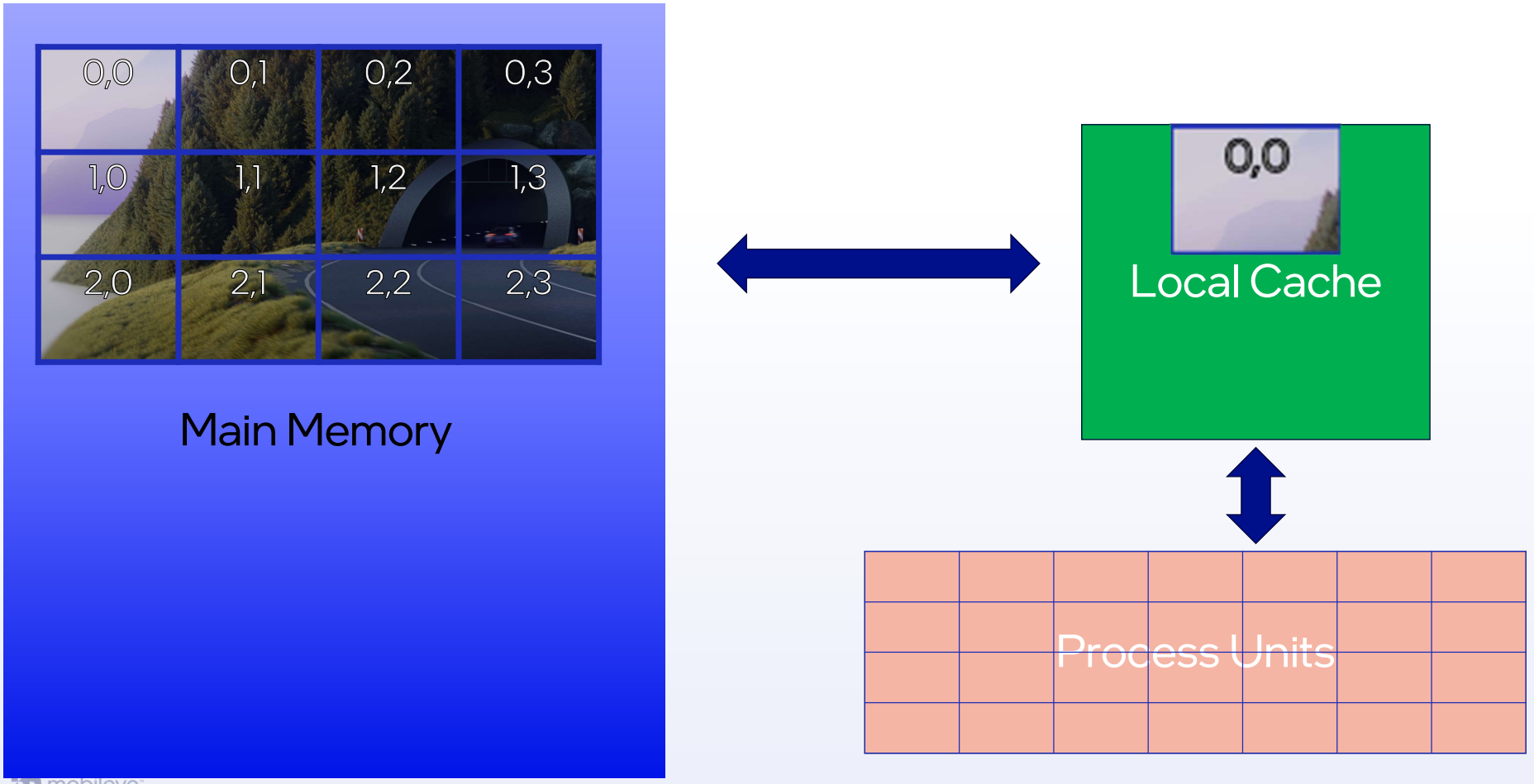
What is tiling all about?



Main Memory



What is tiling all about?



What is tiling all about?

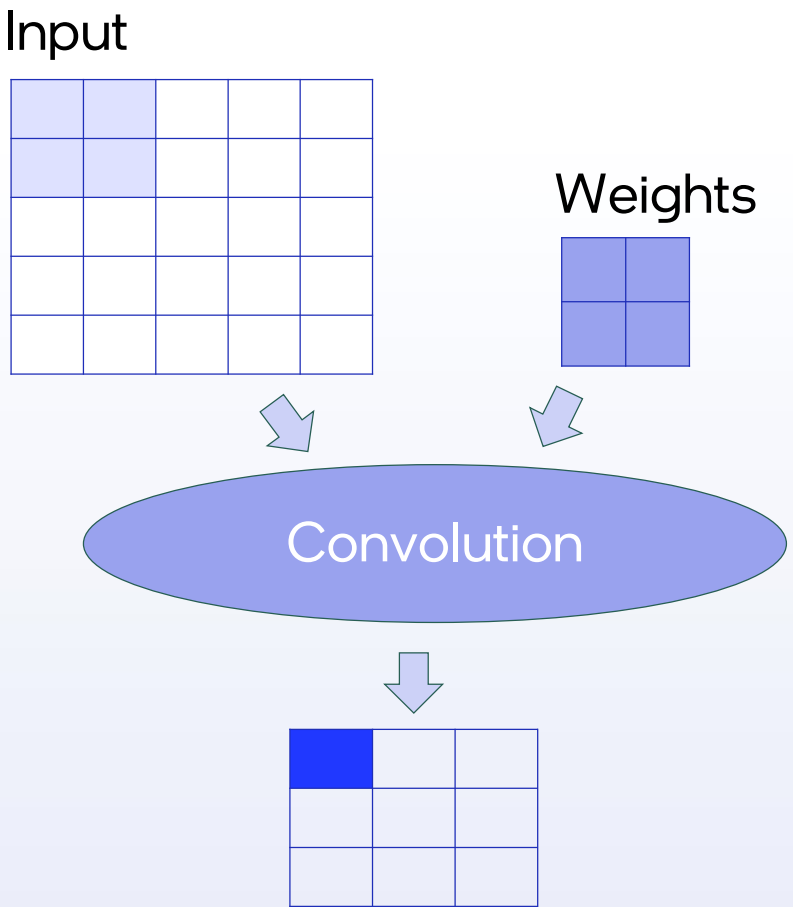
What is tiling all about?

Goal – Reach High Performance

- How?
  - Optimize cache usage by keeping data localized.

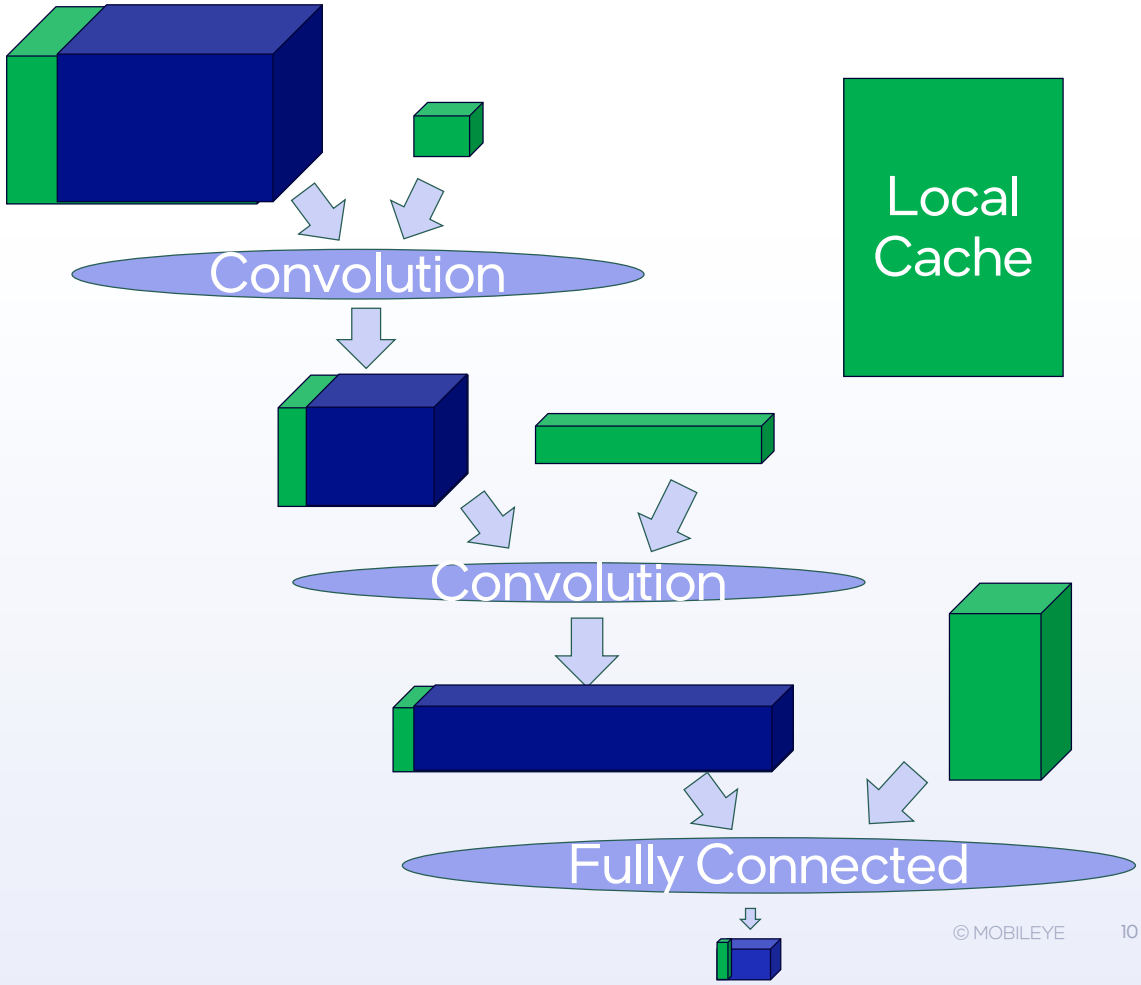
Goal – Reach High Performance

- How?
  - Optimize cache usage by keeping data localized.
  - Retain essential data within the cache for all tiles.



Goal – Reach High Performance

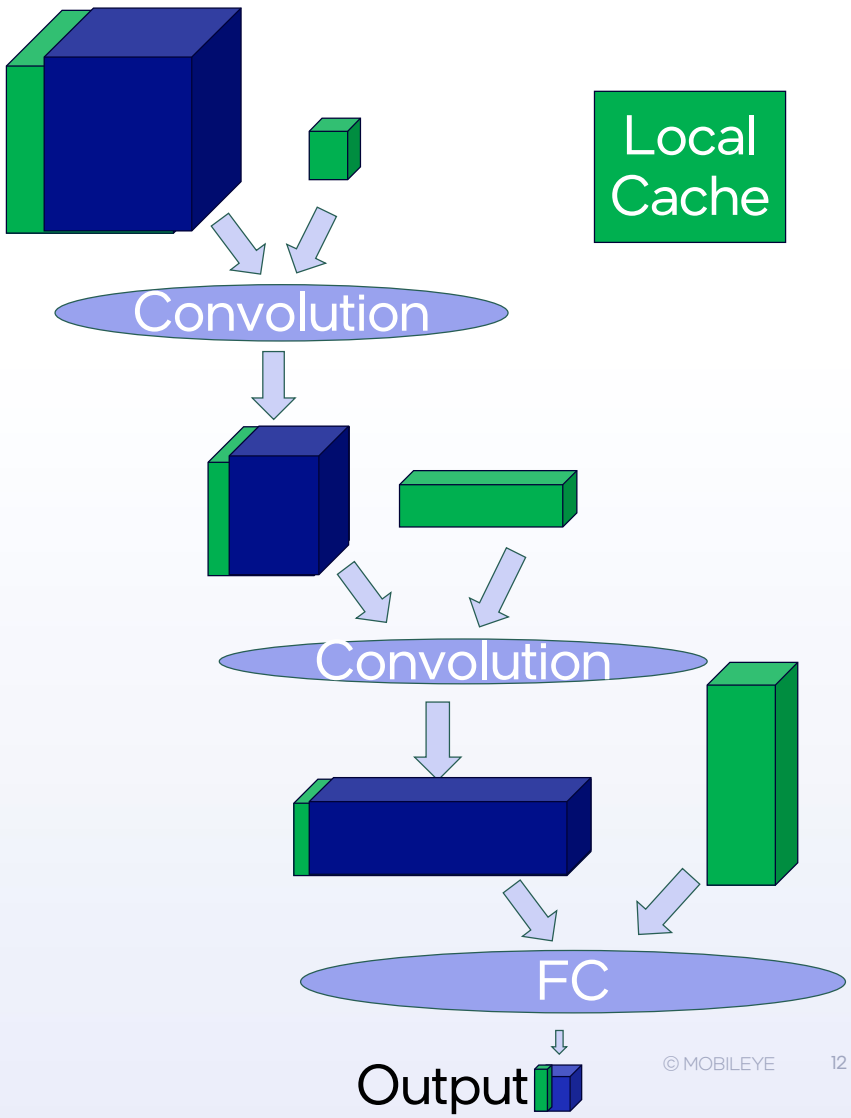
- How?
  - Optimize cache usage by keeping data localized.
  - Retain essential data within the cache for all tiles.
  - Utilize larger tiles to minimize control flow overhead.



Fortunately, transform already exists!

Tile & Fuse - The transform

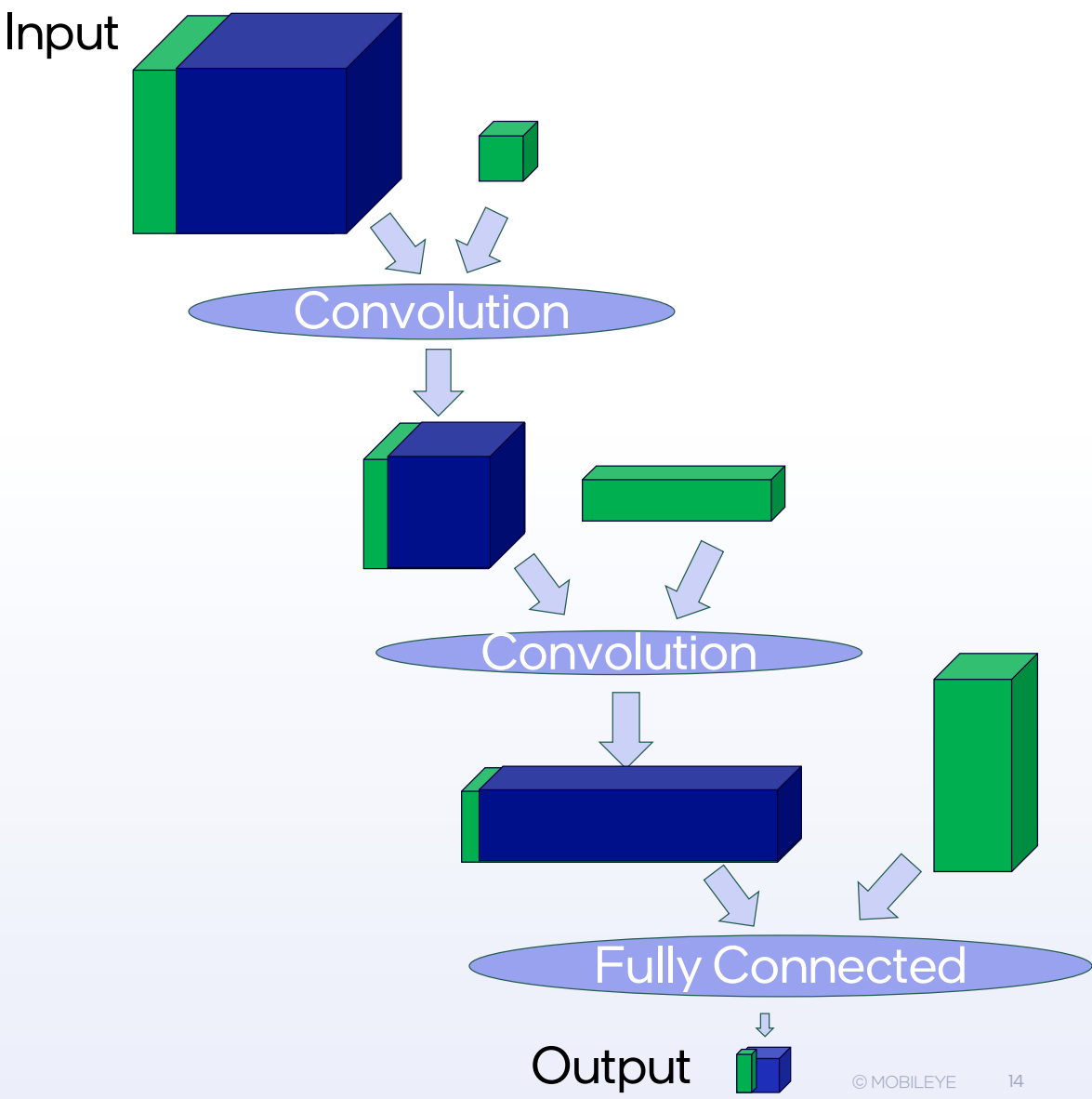
- Demands Tiling Interface.
- General flow:
  - Tile a root operation into loops.
  - Fuse consumer/producer operations into the existing loops.
- Can be controlled by a control function.



How to the select the tile size?

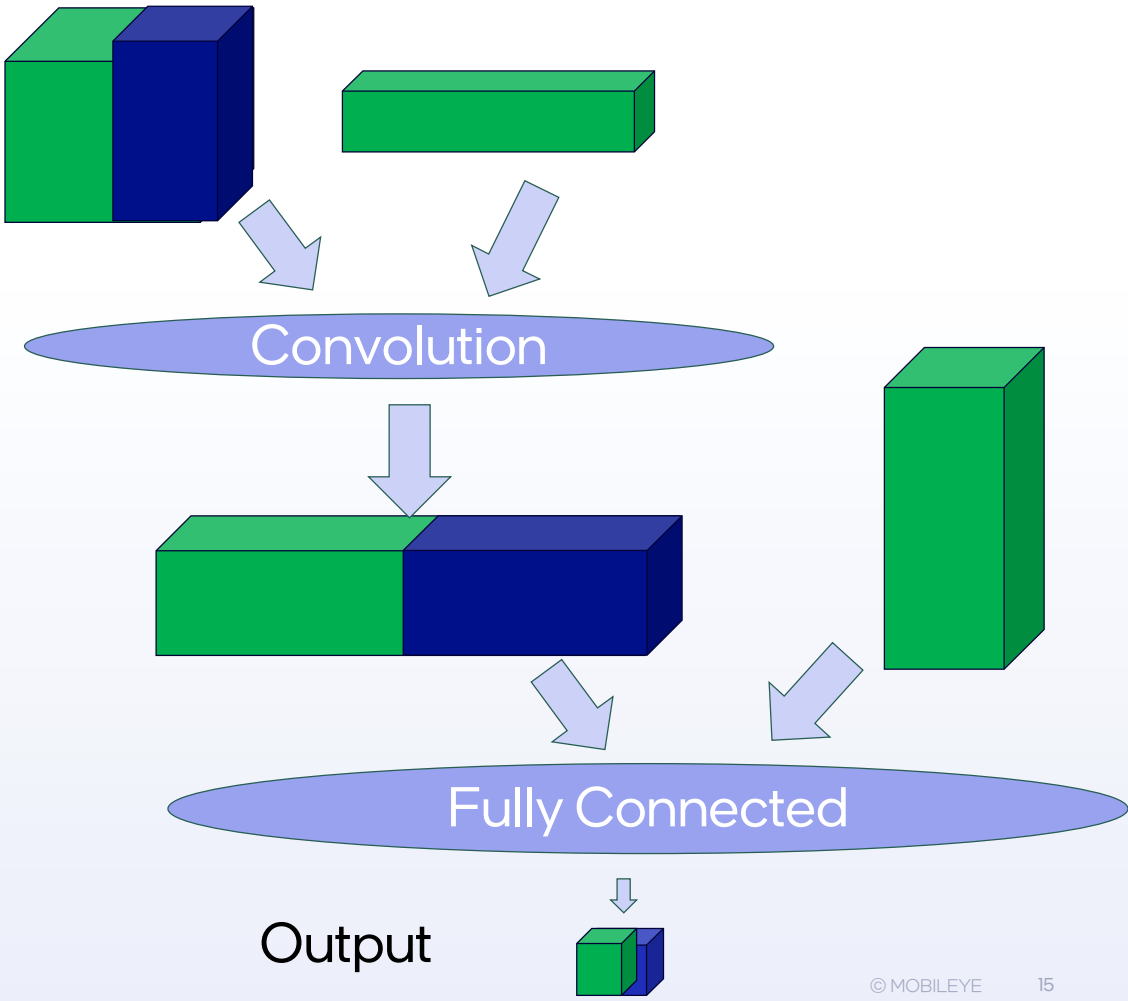
Vertical approach

Increase number of fused operations over tile size



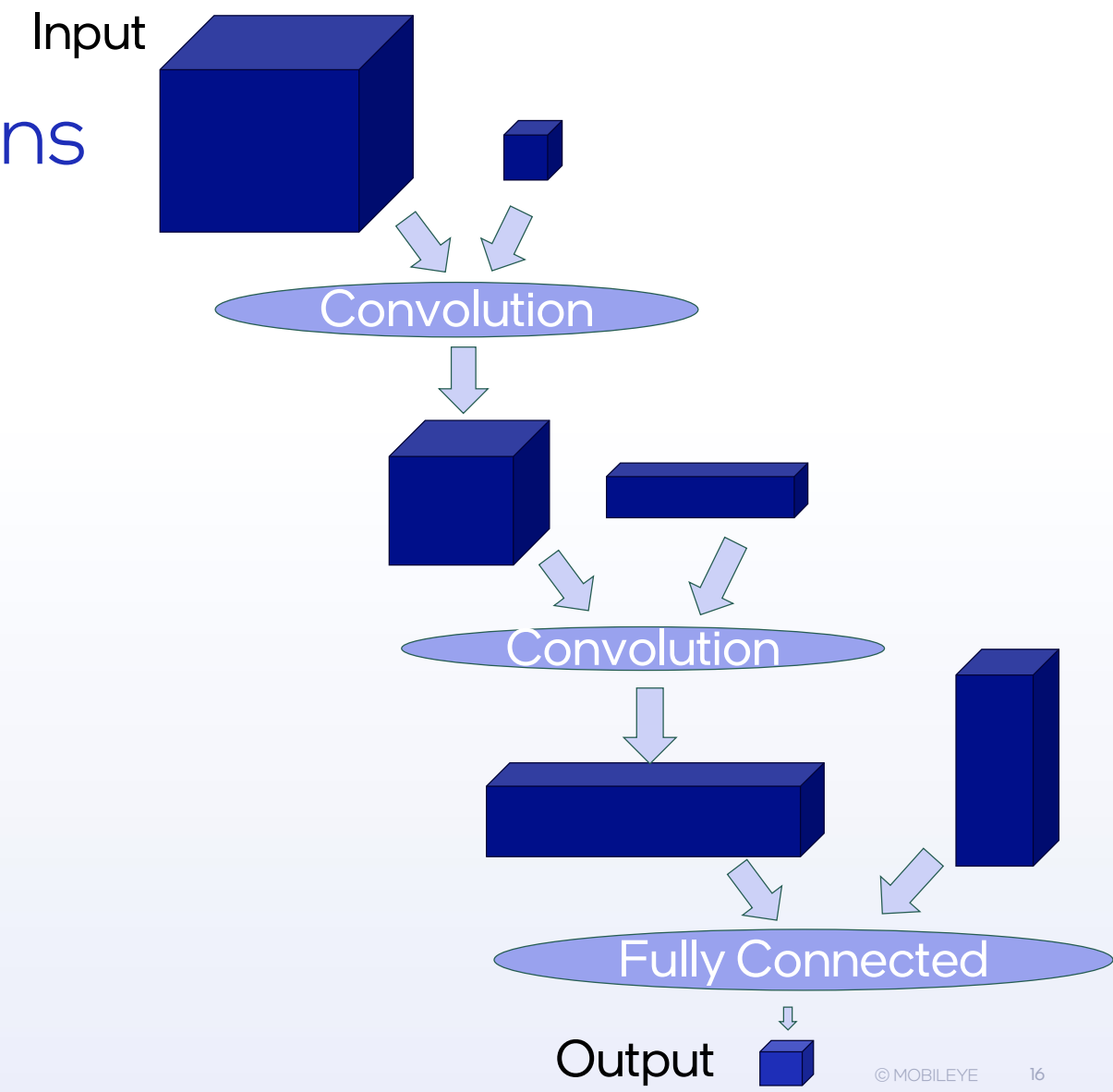
Horizontal Tiling

Increase tile size over number of fused operations



Planning Considerations

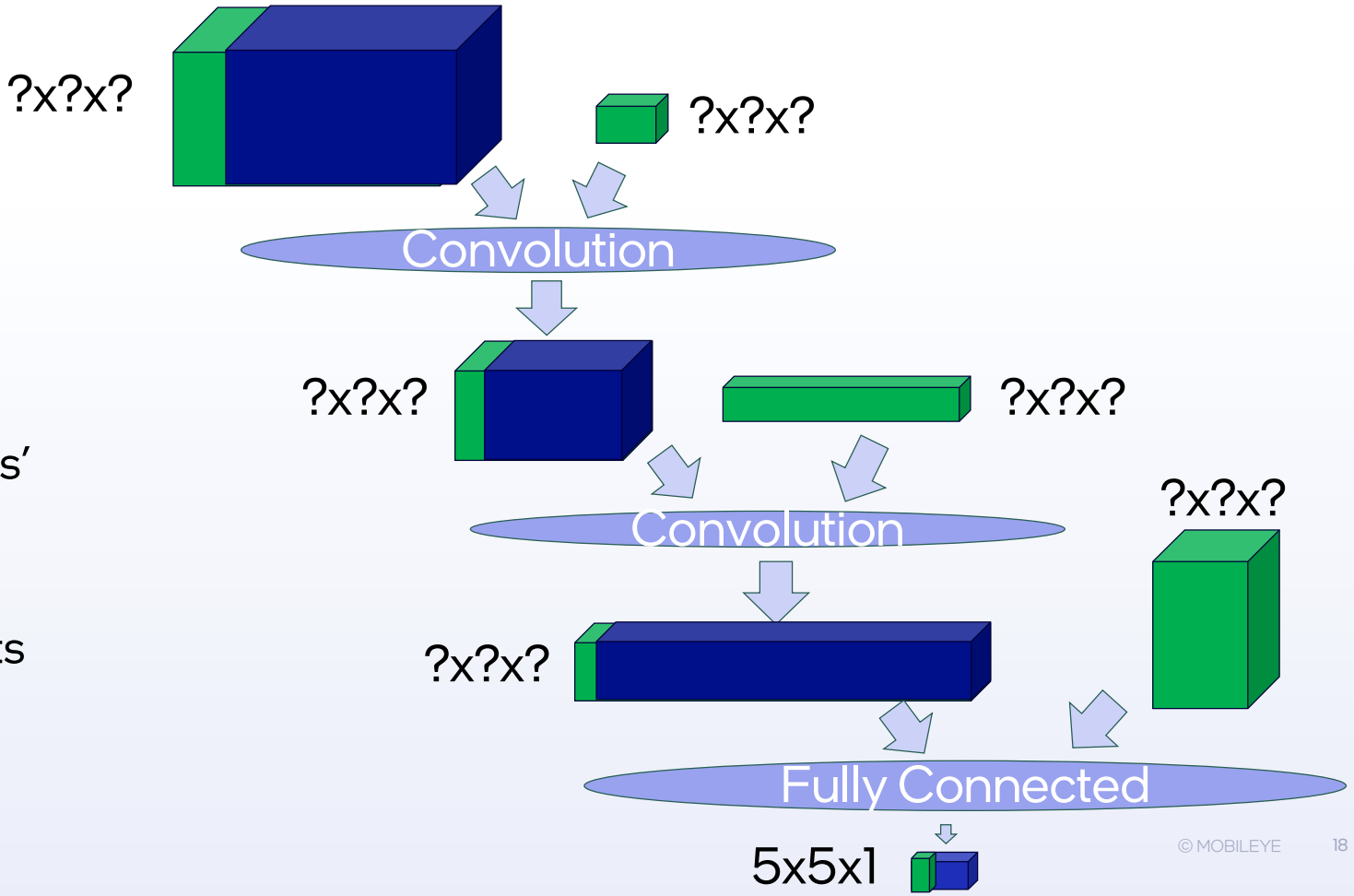
- Find the sweet spot between tile size and fusion to minimize bandwidth
- Overlapping tiles
- Scheduling approaches
- Too big control flow



So, what is missing?

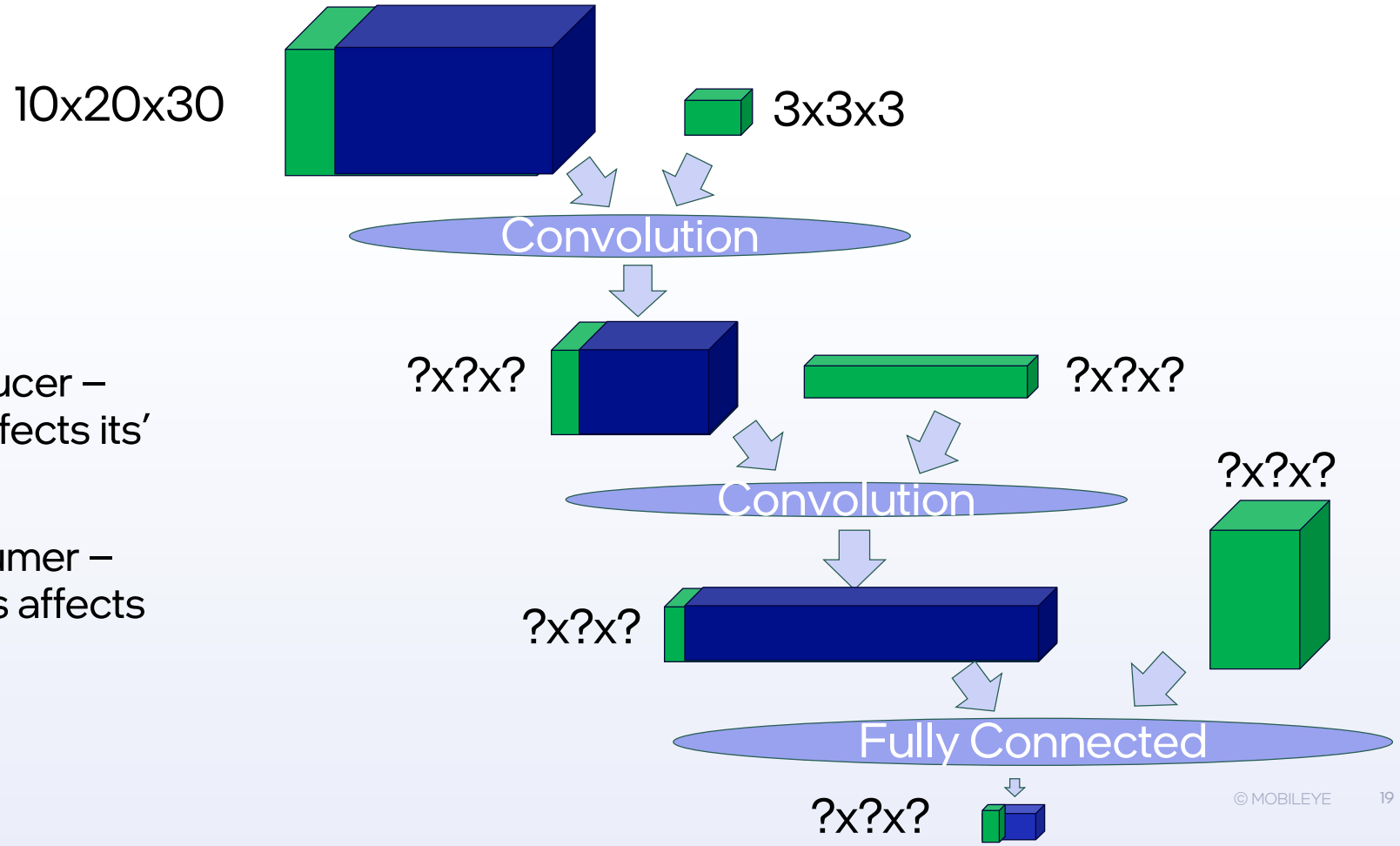
Proposal – Fusion Interface

- Given tiled producer – how its' results affects its' consumers?
- Given tiled consumer – how its' operands affects its' producers?



Proposal – Fusion Interface

- Given tiled producer – how its' results affects its' consumers?
- Given tiled consumer – how its' operands affects its' producers?



RFC - Fusion Analysis Interface for Compute Operations

- **Additional method can be added, let's enhance the interface together!**

- For more details follow the RFC:

https://discourse.llvm.org/t/fusion-analysis-interface-for-compute-operations/85743

Thank you!

Aviad Cohen
Aviad.cohen2@mobileye.com

mobileye™

Developers' Meeting

BERLIN 2025

BOLT'ED CLANG: HOW GOOD IS IT ON AARCH64?

ELVINA YAKUBOVA, SJOERD MEIJER

EUROLLVM 2025

HOW GOOD IS BOLT ON AARCH64?

Questions we wanted to investigate:

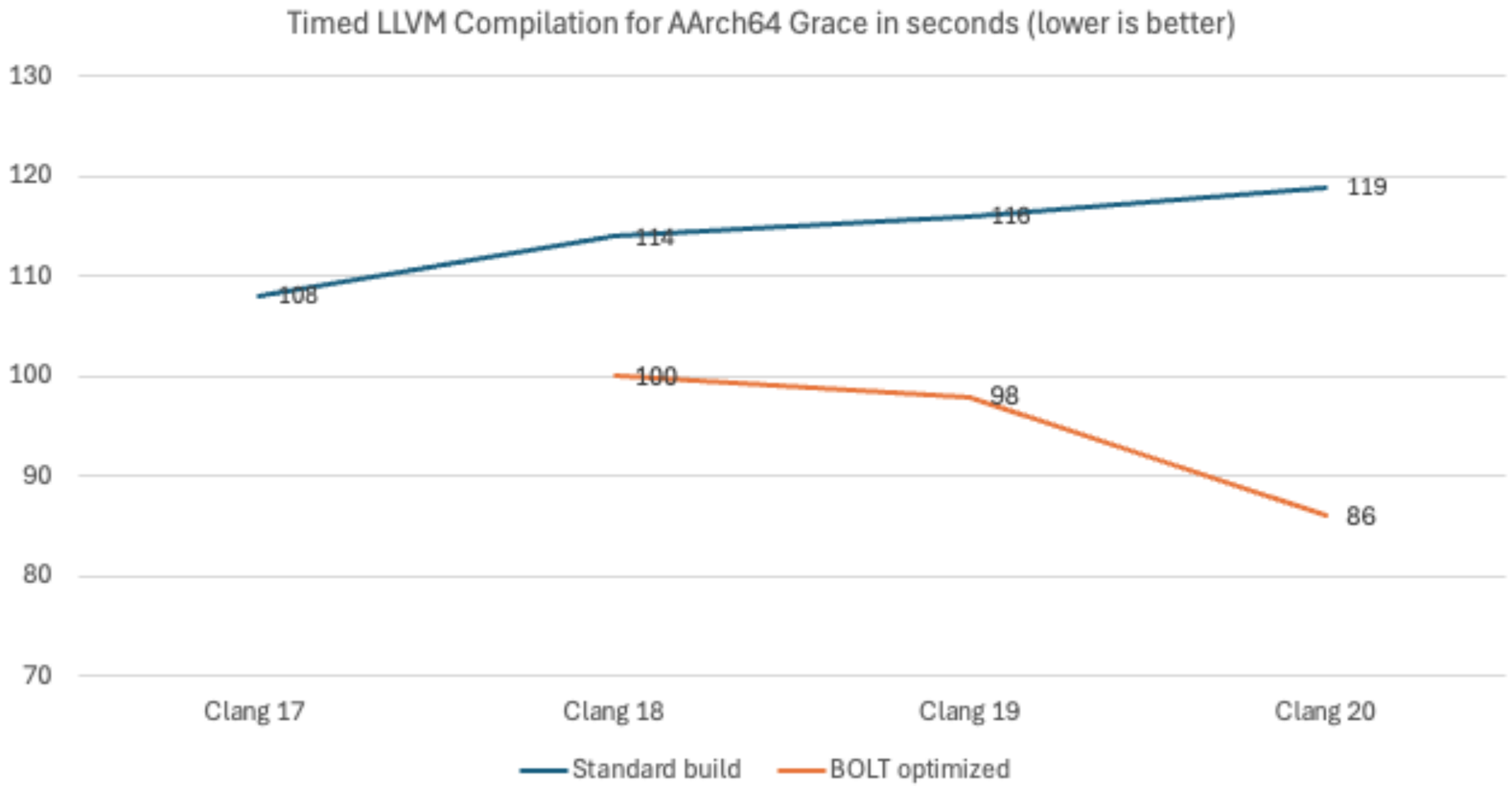
1. What performance improvements can we obtain by using BOLT on AArch64 platforms?
2. Should we BOLT our Clang build?
3. And can we do better and change build process?

We will show:

- Performance results for timed compilation of LLVM and CTMark (LLVM test-suite)
- The trends for Clang-18, Clang-19, and Clang-20
- Show experiments with more and different profiles

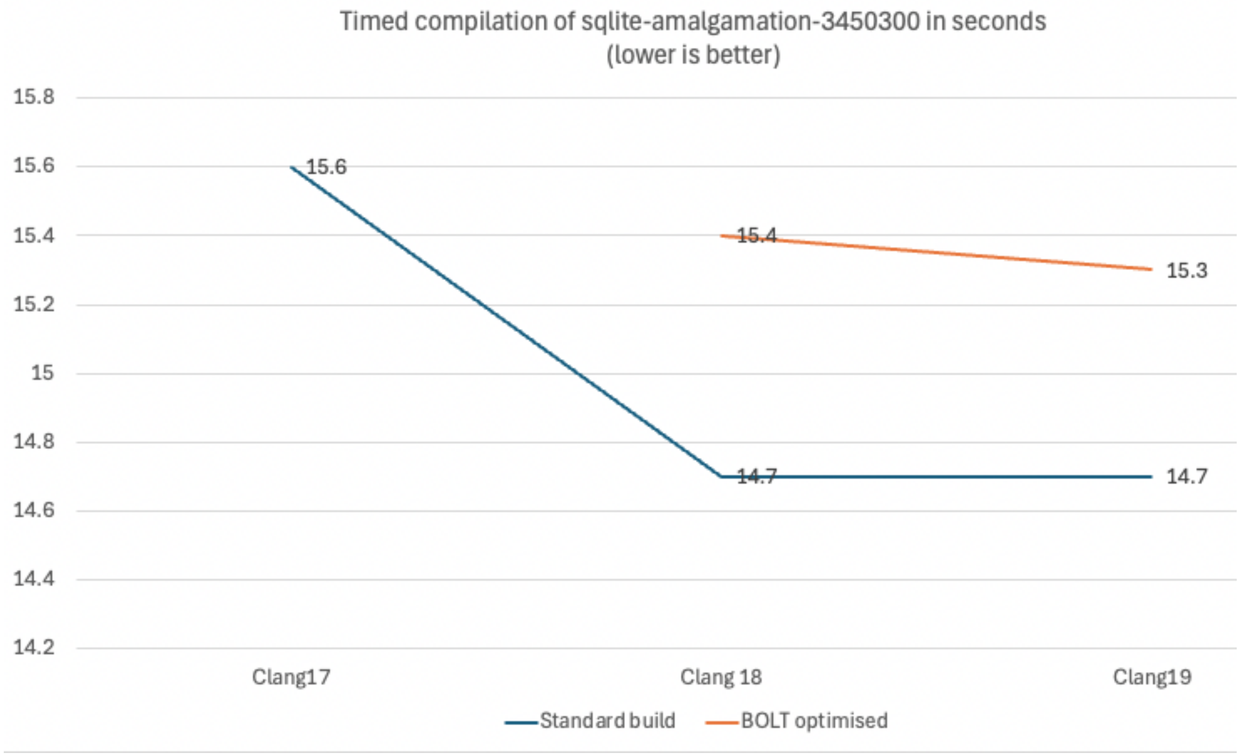
BOLT'ED CLANG: LTO/PGO/BOLT OPTIMISED COMPILER

- BOLT optimised AArc64 Clang toolchain
  - A.k.a.: how do we create a fast compiler?
  - Metric: measure compilation time of the BOLT'ed compiler: 27% speed-up of BOLT'ed Clang-20 compared to a standard build:



IS IT GOOD FOR ALL WORKLOADS?

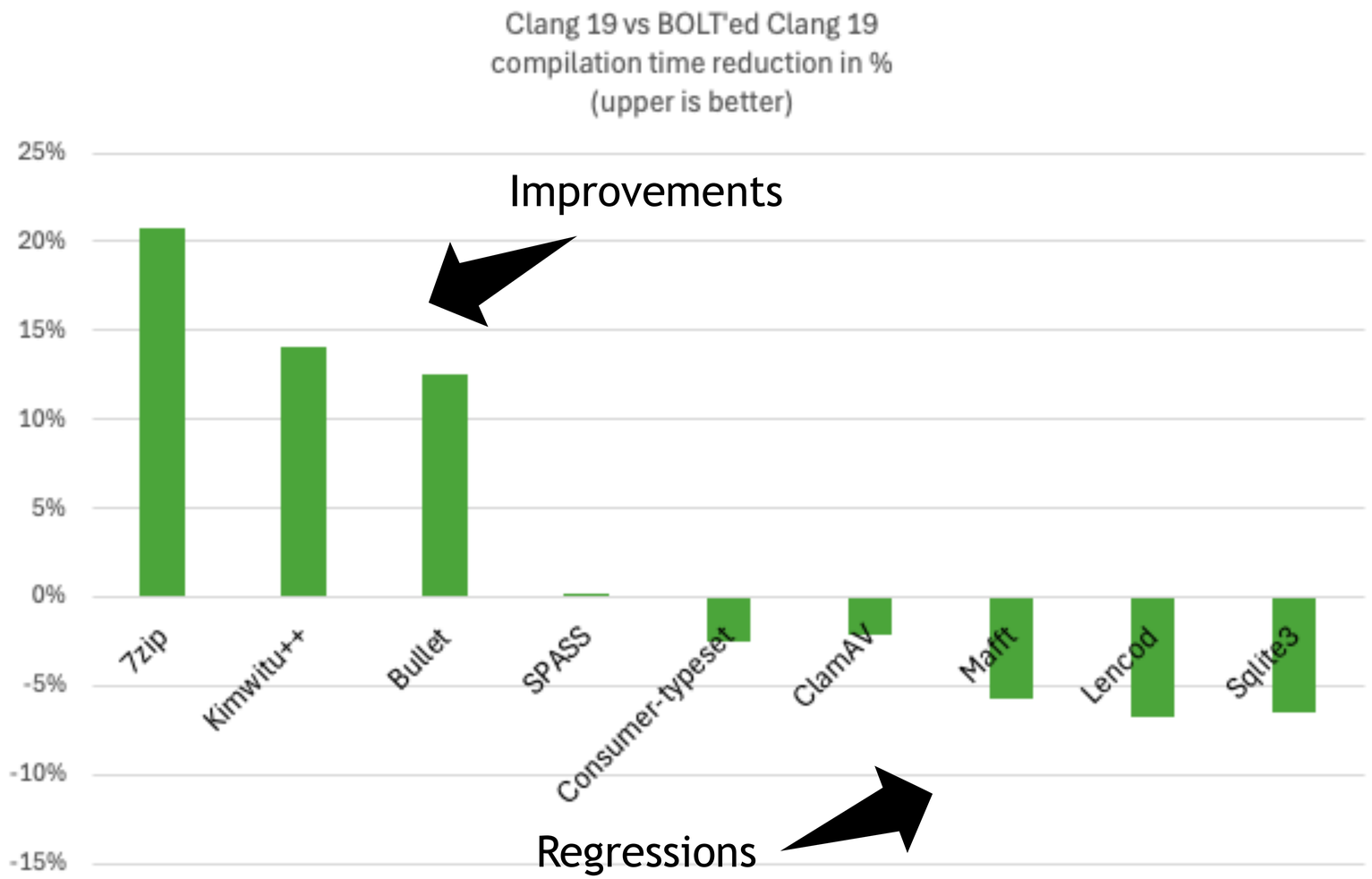
- 4% compile-time regression for SQLite with BOLT'ed Clang:



Hypothesis:

- BOLT'ed Clang is trained on a modern C++ code (LLVM),
- Maybe this works less well for SQLite that is C-code.

IT'S NOT ALL GOOD, MORE REGRESSIONS...

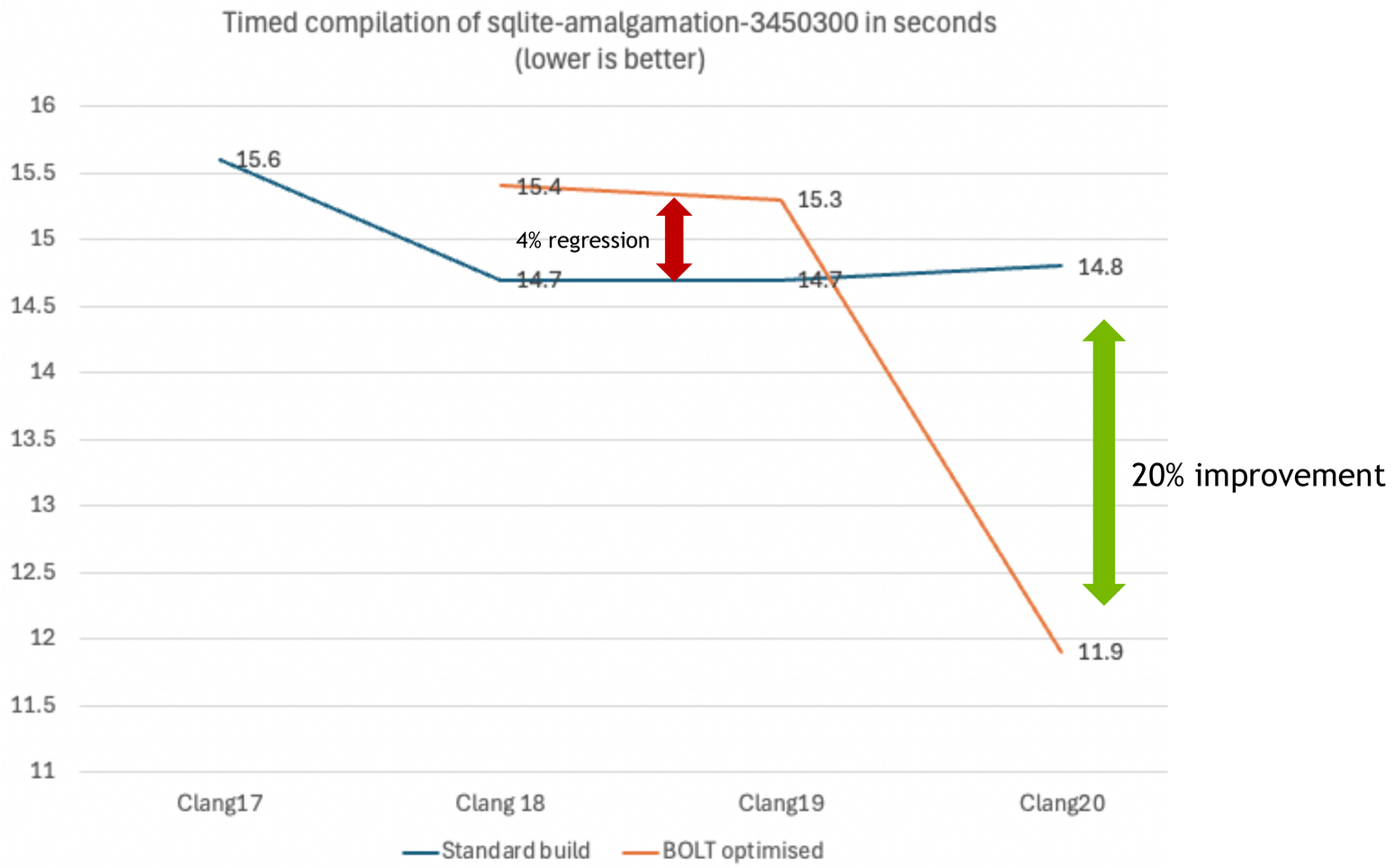


CTMark C-code base:

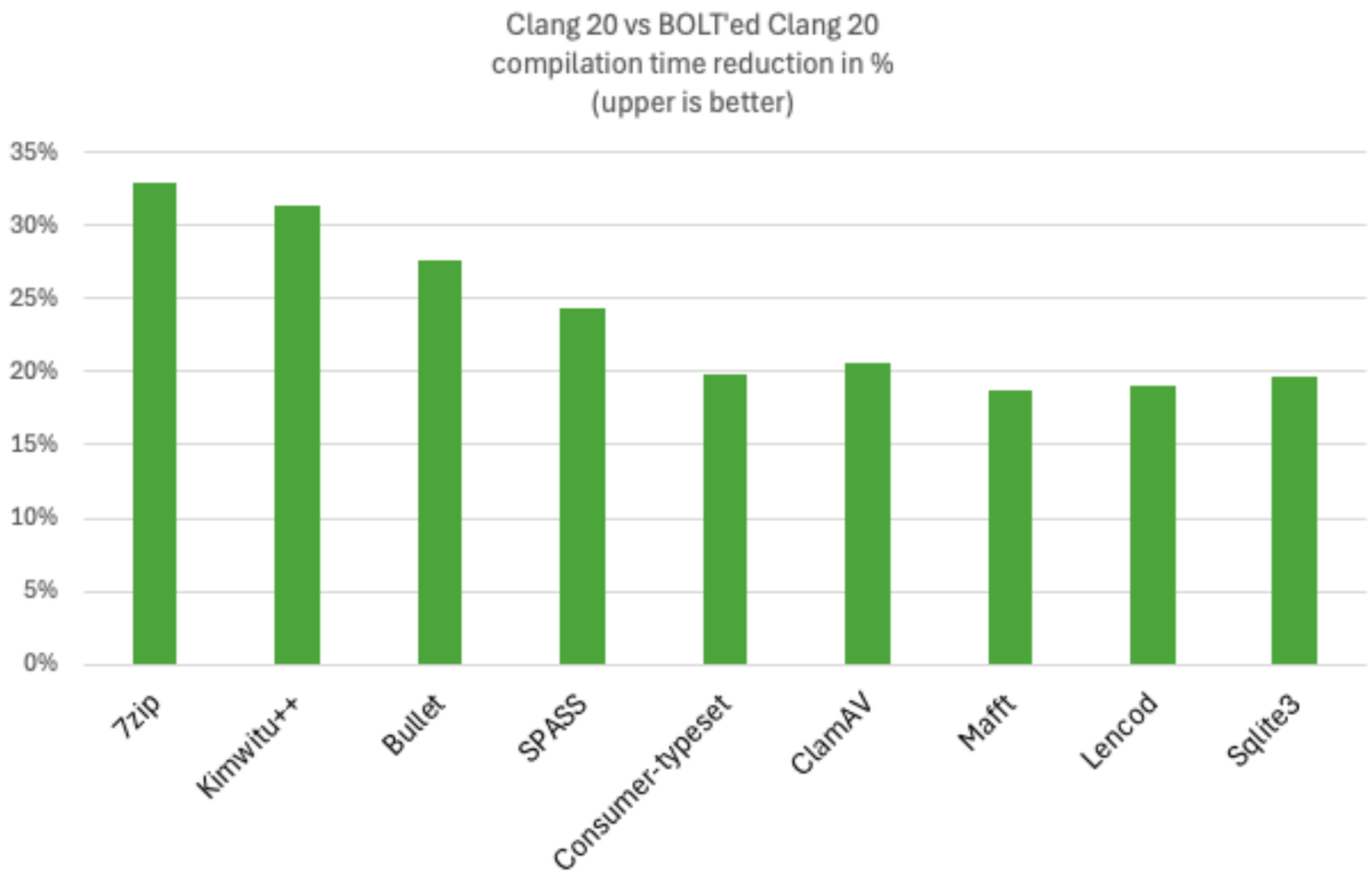
- SPASS
- Consumer-typeset
- ClamAV
- Mafft
- Lencod
- Sqlite

CLANG 20: GAME CHANGER

- Massive 20% SQLite compile-time improvement!



ONLY IMPROVEMENTS WITH CLANG 20



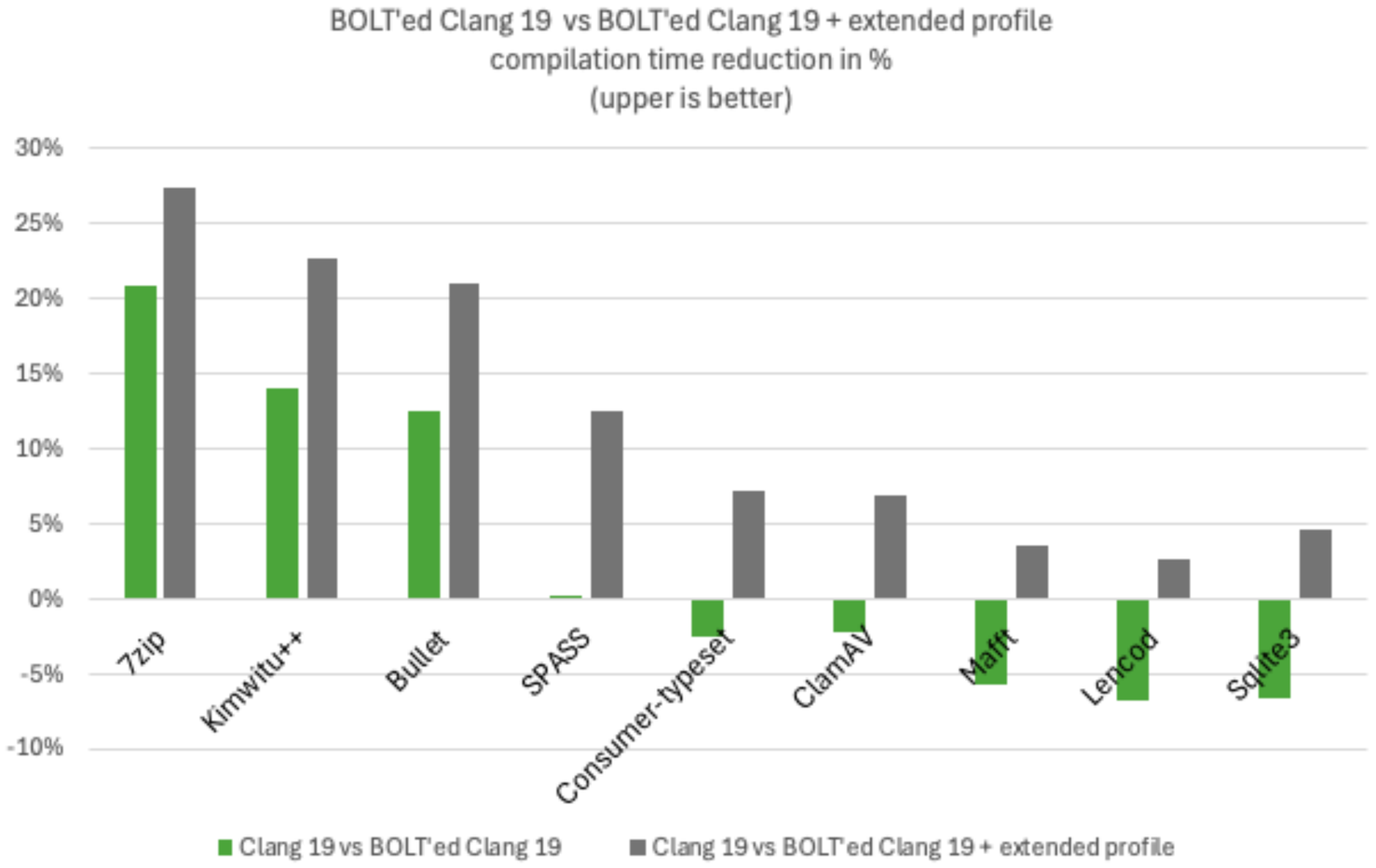
CAN WE DO BETTER?

- BOLT'ed Clang-20 improvements could come from:
  - CMake configurations: different options passed on,
  - BOLT learned new optimisations,
  - More/better profiles?

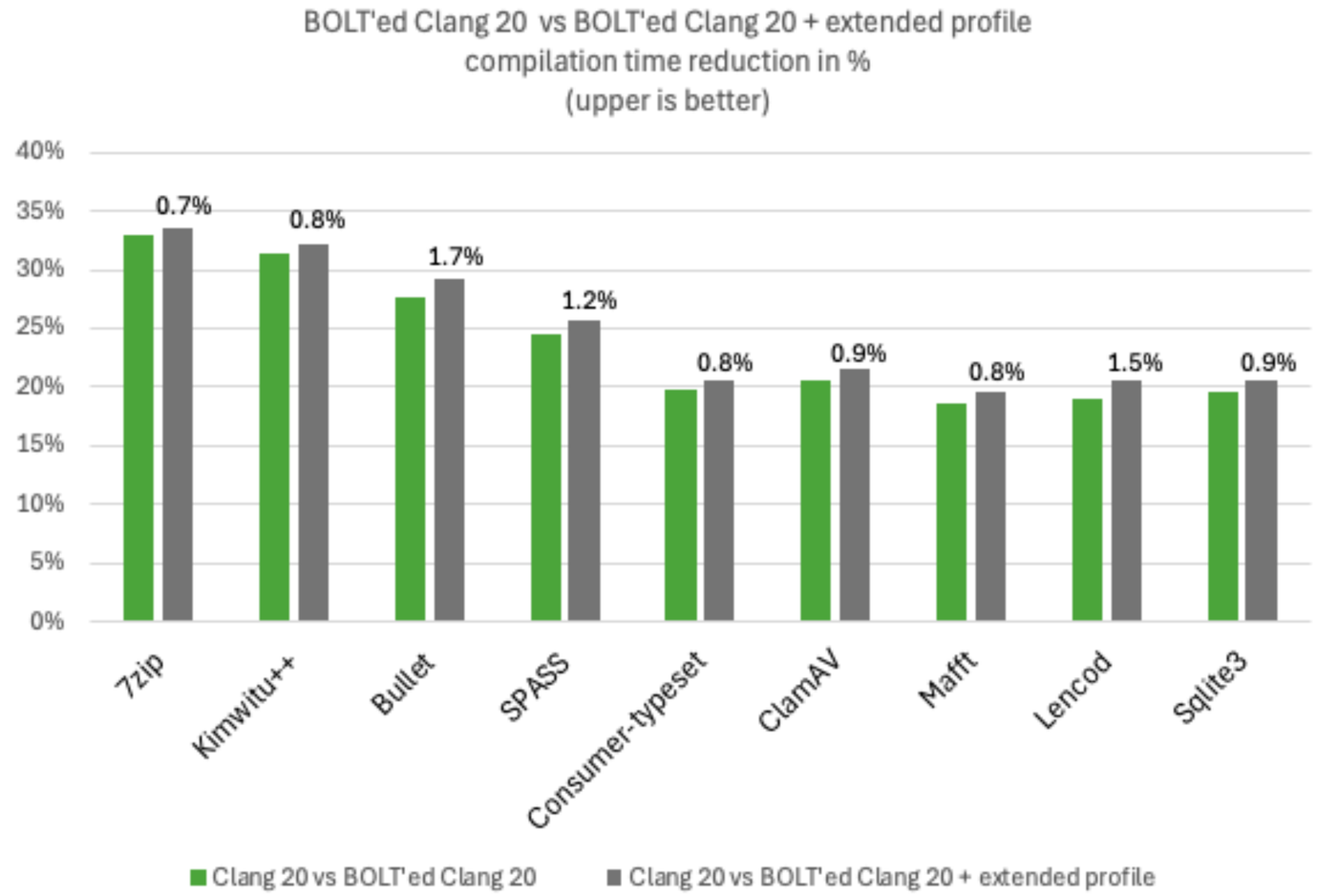
- Can we do better?
  - BOLT'ed Clang is trained on a modern C++ code-base (LLVM),
  - Should we extend the training stage with more/different profiles?

- Extended Profile:
  - Collect profiles for CTMark from the LLVM test-suite (C code-base),
  - Collect profiles for LLVM (C++ code-base),
  - Merge LLVM + CTMark profiles and use that as input to BOLT.

MORE PROFILES FIX THE CLANG 19 RESULTS

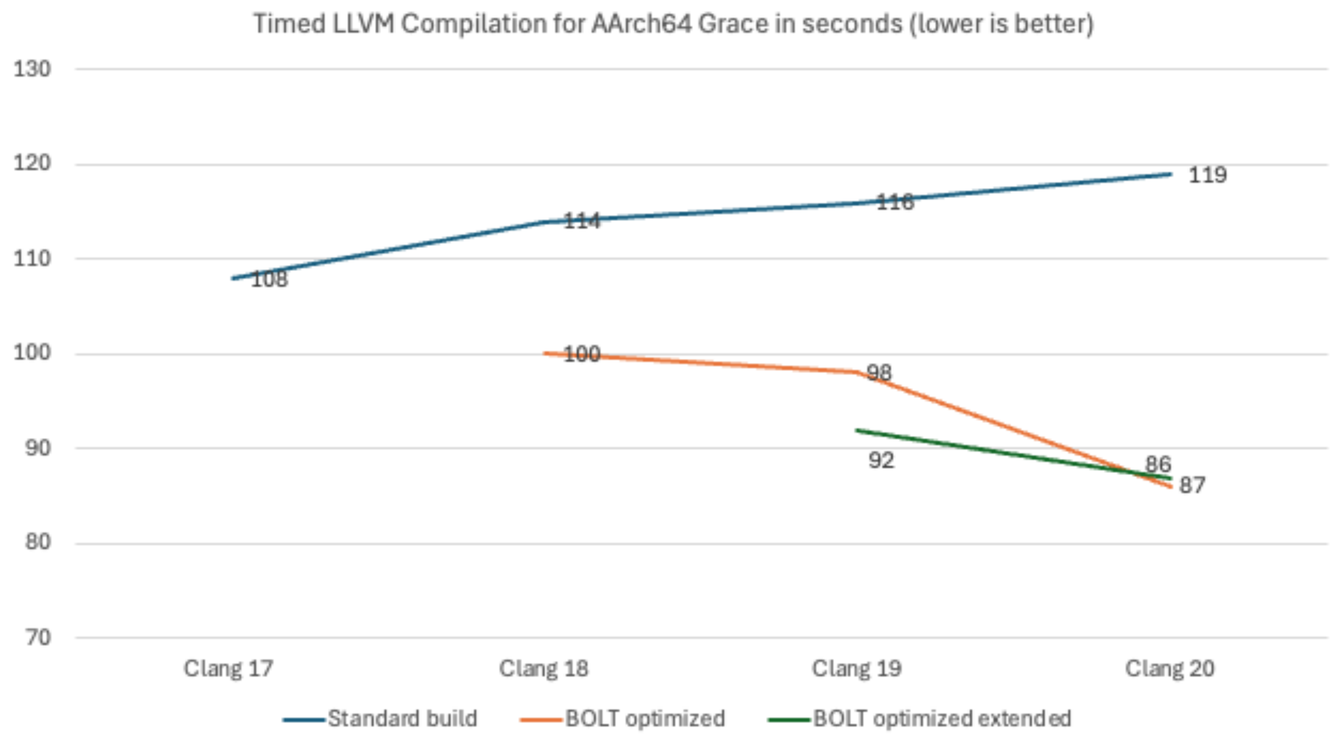


EXTENDED PROFILES: ADDITIONAL IMPROVEMENTS LESS THAN 2%



LTO/PGO/BOLT OPTIMISED CLANG COMPILER

- Additional 6% improvement on Clang 19
- No difference on Clang 20



CONCLUSIONS

- Clang-20 and BOLT-20 are great, also on AArch64!
  - BOLT'ed Clang-20 is 27% faster than Clang-20
  - Universally good: the same or better performance (for LLVM and CTMark)
  - Fixes the issues with Clang-19 and older versions that were not so great yet.

- Yes, Clang releases should be BOLT'ed
  - They started with the latest release

- Extended profiles:
  - Clang-20 is now also trained on libLLVMSupport: big improvement
  - Training it even more with CTMark: almost makes no difference!
  - The current CMake Clang/BOLT build process and configuration is enough

- Future work:
  - CTMark apps are small, investigate more/bigger apps,
  - With extended profiles, they should also be added to the PGO stage (i.e. not only BOLT)


Thank you
NVIDIA

Developers' Meeting

BERLIN 2025

EuroLLVM 2025 - Berlin

MLIR Tensor Compiler

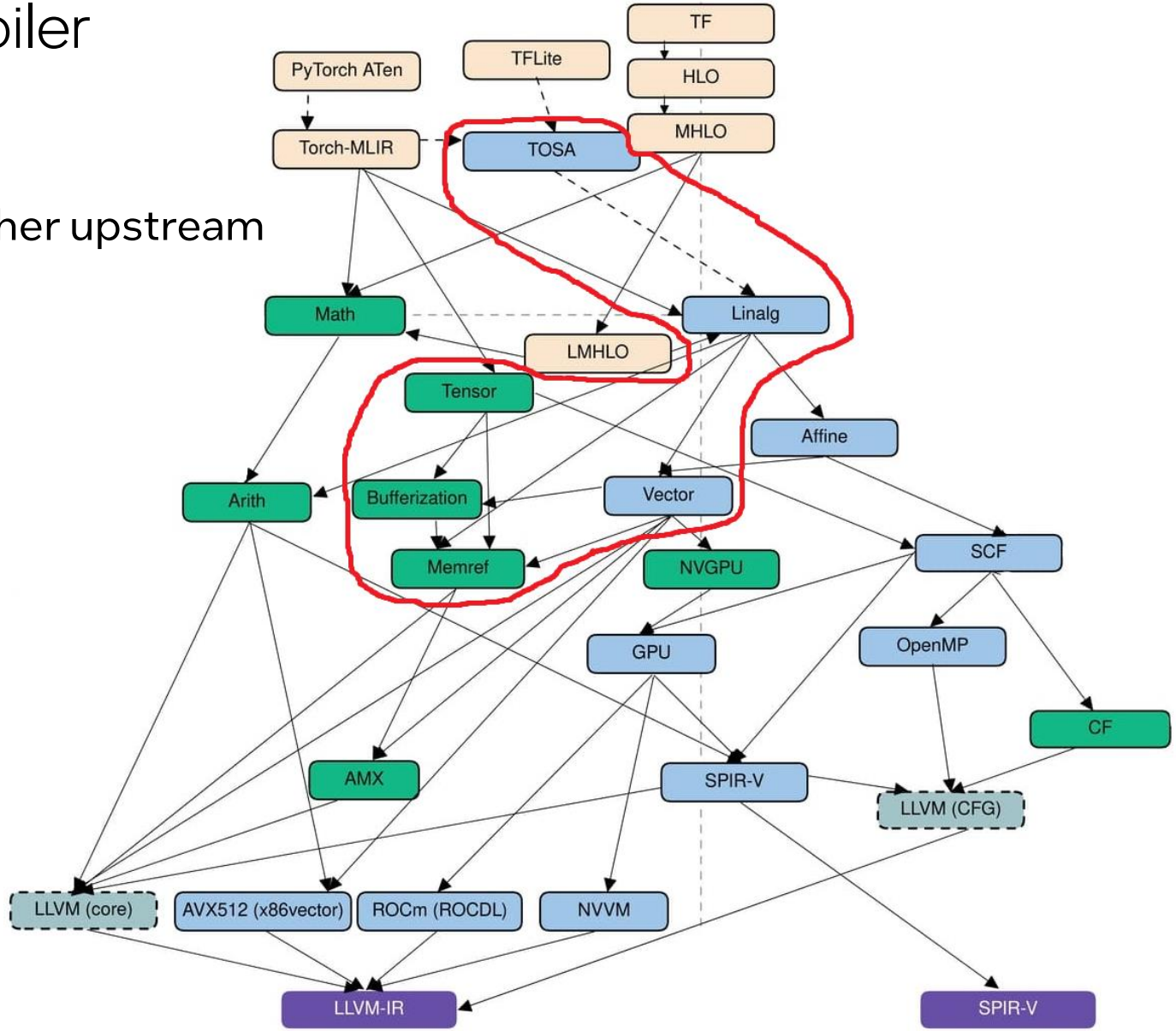
design group & charter update

Rolf Morel & Renato Golin

Upstream MLIR Tensor Compiler

- MLIR-based toolkit for ML compilers
  - Great value in developing this together upstream

- Main dialects:
  - linalg
  - tensor
  - memref
  - vector
  - bufferization
  - tosa



Original image credit: Quinn Dawkins & Alex Zinenko

MLIR organization – dialect groupings

- Originates with MLIR Governance efforts from last year
  - [RFC] MLIR Project Charter and Restructuring by Renato Golin, Stella Laurenzo, Chris Lattner, Alex Zinenko, Jacques Pienaar, Mehdi Amini
  - [Survey] MLIR Project Charter and Restructuring Survey by Renato
  - MLIR Organization & Charter by Renato, Stella, Alex, Jacques, Chris, and Andrzej Warzynski , Nicolas Vasilache, Mahesh Ravishankar

MLIR organization – dialect groupings

- Originates with MLIR Governance efforts from last year
  - [RFC] MLIR Project Charter and Restructuring by Renato Golin, Stella Laurenzo, Chris Lattner, Alex Zinenko, Jacques Pienaar, Mehdi Amini
  - [Survey] MLIR Project Charter and Restructuring Survey by Renato
  - MLIR Organization & Charter by Renato, Stella, Alex, Jacques, Chris, and Andrzej Warzynski , Nicolas Vasilache, Mahesh Ravishankar

- Tensor Compiler dialect grouping
  - **linalg**, **tensor**, **memref**, **vector**, **bufferization**, **tosa**
  - Own community of stakeholders
  - In need of an overarching charter
  - Clear governance: upstream consensus

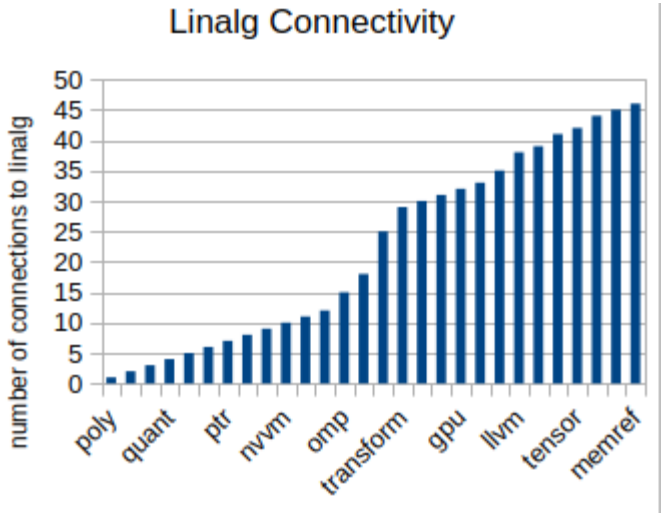
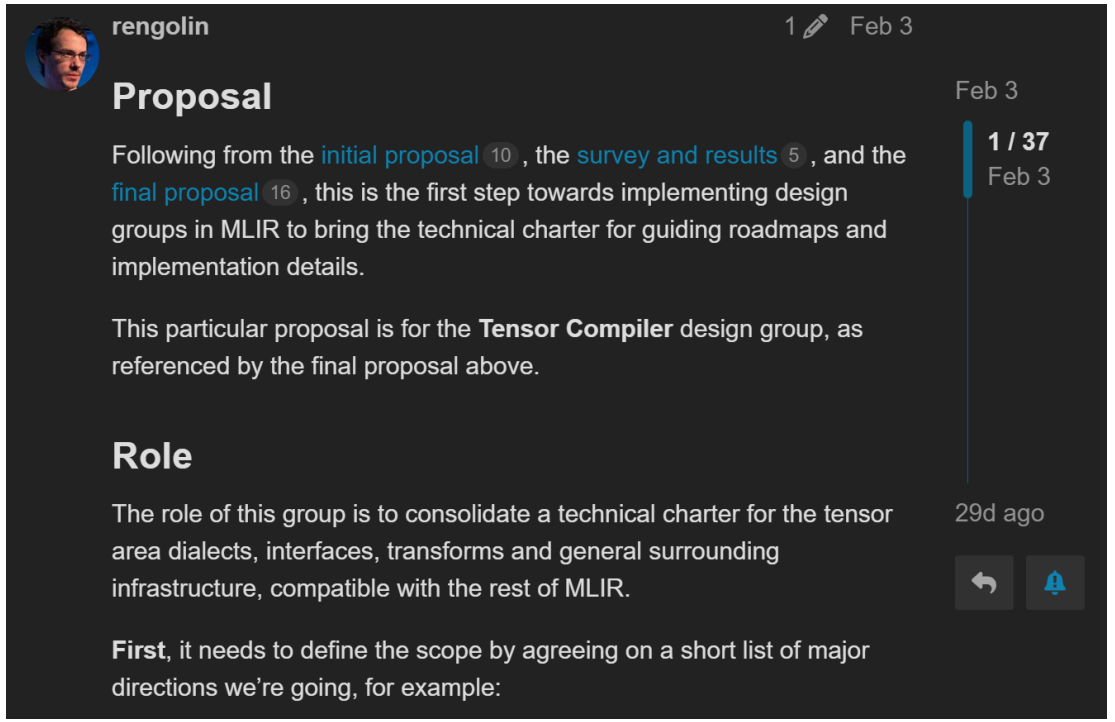


Image credit: Renato Golin

Tensor Compiler Design Group

- MLIR Tensor Compiler Design Group by Renato

  Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure



Tensor Compiler Design Group

- MLIR Tensor Compiler Design Group by Renato

  Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

  Scope

  – linear algebra – tensor semantics – bufferization – memref semantics – vector semantics

  – canonicalization (op aliasing within linalg) – ingress & egress – dialect design – flows

Tensor Compiler Design Group

- MLIR Tensor Compiler Design Group by Renato

  Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

  Scope

  – linear algebra – tensor semantics – bufferization – memref semantics – vector semantics

  – canonicalization (op aliasing within linalg) – ingress & egress – dialect design – flows

  Documentation updates

  – Consolidate conflicting roadmaps

  – Update outdated rationale docs

  – Start an overarching charter

Tensor Compiler Design Group

- MLIR Tensor Compiler Design Group by Renato

  Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

  Scope

  – linear algebra – tensor semantics – bufferization – memref semantics – vector semantics

  – canonicalization (op aliasing within linalg) – ingress & egress – dialect design – flows

  Documentation updates

  – Consolidate conflicting roadmaps

  – Update outdated rationale docs

  – Start an overarching charter

  Infrastructure for distributed usage

  – Devise a path for more flexibility for users (also across upstream projects)

  – E.g., easier dialect extensions, canonicalized transform requirements, better coverage

Tensor Compiler Design Group – Members

Volunteers from active contributors and representative stakeholders

Alex Zinenko Brium	Renato Golin Intel	Jacques Pienaar Google	Matthias Springer Nvidia
Quinn Dawkins AMD	Javed Absar Qualcomm	Jakub Kuderski AMD	Suraj Sudhir Arm
Rolf Morel Intel	Andrzej Warzynski Arm	Diego Caballero Nvidia	Kunwar Grover AMD

Tensor Compiler Design Group – Members

Volunteers from active contributors and representative stakeholders

Alex Zinenko Brium	***Renato Golin*** Intel	***Jacques Pienaar*** Google	***Matthias Springer*** Nvidia
Quinn Dawkins AMD	Javed Absar Qualcomm	Jakub Kuderski AMD	Suraj Sudhir Arm
Rolf Morel Intel	Andrzej Warzynski Arm	Diego Caballero Nvidia	Kunwar Grover AMD

In bold: MLIR Area Team, a distinct governance effort

Tensor Compiler Design Group – Members

Volunteers fr

Alex Zinenko
Brium

Quinn Dawkins
AMD

Rolf Morel
Intel

atthias Springer
Nvidia

Suraj Sudhir
Arm

Kunwar Grover
AMD

Modular

Democratizing AI Compute, Part 8:
What about the MLIR compiler infrastructure?

CHRIS LATTNER

A New Hope: Improved MLIR Governance

The tensions have simmered for years—and they're deeply felt across the broader LLVM and MLIR communities.

Fortunately, **there's a new hope**: LLVM is a meritocratic community with a long track record of aligning engineers—even when their companies are at war in the market. The MLIR community is filled with amazing engineers who have poured years of their hearts and souls into improving the project to work through these challenges, and progress is now happening!

MLIR now has a new **Area Team** to help guide its evolution, along with a **new organizational structure and charter** and **governance group**. The charter defines separate area groups: MLIR Core (the domain-independent infrastructure), and the dialects (like the machine learning-specific pieces). I am extremely thankful to everyone who is spending time to improve MLIR and work through these issues—such work has a profound impact on everyone building into the ecosystem as well as the downstream users.

In bold: MLIR Area Team, a distinct governance effort

Tensor Compiler Design Group – Modus Operandi

- Regular meetings
  - Bi-weekly group meetings
    - Agenda & notes: MLIR Tensor Compiler Design Group - Meetings
    - … only had three (3) so far
  - First **O**pen **D**esign **M**eeting scheduled late April (date TBC)

Tensor Compiler Design Group – Modus Operandi

- Regular meetings
  - Bi-weekly group meetings
    - Agenda & notes: MLIR Tensor Compiler Design Group - Meetings
    - … only had three (3) so far
  - First **O**pen **D**esign **M**eeting scheduled late April (date TBC)
- In practice, members bring in-the-pipeline, community-relevant topics
  - In essence *pre-RFCs* — workgroup provides space for quick iteration
    - Rapid top-of-mind responses for low overhead feedback
  - Next: posted as RFC, with same consensus process as any other RFC

Tensor Compiler Design Group – Modus Operandi

- Regular meetings
  - Bi-weekly group meetings
    - Agenda & notes: MLIR Tensor Compiler Design Group - Meetings
    - … only had three (3) so far
  - First **O**pen **D**esign **M**eeting scheduled late April (date TBC)
- In practice, members bring in-the-pipeline, community-relevant topics
  - In essence *pre-RFCs* — workgroup provides space for quick iteration
    - Rapid top-of-mind responses for low overhead feedback
  - Next: posted as RFC, with same consensus process as any other RFC
- All workgroup-generated documentation is public

MLIR Tensor Compiler Design Group - Overview document

Tensor Compiler Design Group – Progress on Vector

MLIR Tensor Compiler Design Group - Vector Dialect: Refactoring + Re-design ideas

RFCs which benefitted from live discussion:

- [RFC] Allow pointers as element type of `vector`
  - Upshot: VectorElementTypeInterface with semantics of only allowing “atomic” elements
- [RFC] Improving gather codegen for Vector Dialect
  - Addresses abstraction gap which lead to early loss of structured indexing
- [RFC] Generalize tiling to operate on ShapedType
  - Proposes extending infrastructure for tiling/blocking beyond linalg-on-tensor/memref
- In-the-pipeline RFC on reducing references to LLVM LangRef in vector dialect docs

Tensor Compiler – other recent significant changes

- Transpose attribute for Linalg matmul operations
  - Linalg.matmul (and batch_matmul) now have an affine_maps attribute
- Introduce linalg.contract : $D[\mathbf{J}] = \left(\bigoplus_{((I^A \cup I^B) \setminus \mathbf{J})} A[I^A] * B[I^B] \right) \oplus C[\mathbf{J}]$
  - Op generalizing all contraction ops (e.g. matmul variants and matvec and dot and …)
- Extend Linalg elemwise named ops semantics
  - linalg.elementwise with affine_maps replacing linalg.elem_wise_{unary,binary}
- Move `tensor.pack` and `tensor.unpack` into Linalg
  - Fix layering issue; facilitate interactions with linalg ops; allow for packing on memrefs

Topics we are looking to make progress on

1. [RFC] Should we restrict the usage of 0-D vectors in the Vector dialect?
2. Remove ops with overlapping functionality from Vector, e.g. extractelement -> extract
3. Various issues in Vector regarding consistent naming and semantics
4. Pros & cons of splitting vector dialect into high-level and low-level parts
5. Vector vs Tensor types – another go at clarifying their distinctive roles

Topics we are looking to make progress on

1. [RFC] Should we restrict the usage of 0-D vectors in the Vector dialect?
2. Remove ops with overlapping functionality from Vector, e.g. extractelement -> extract
3. Various issues in Vector regarding consistent naming and semantics
4. Pros & cons of splitting vector dialect into high-level and low-level parts
5. Vector vs Tensor types – another go at clarifying their distinctive roles
6. Revisit how to attach layouts to tensors & vectors
7. Enshrine in the charter the significance of projected permutation affine_maps
8. Pick up stalled progress on common-ing up linalg conv ops
9. Compute type on linalg ops given correspondence with imperfect loop nest
10. Op aliasing in linalg, e.g. linalg.matmul vs linalg.contract vs linalg.generic
    - Equiv. class perspective on matching w.r.t. [RFC] Linalg operation tree

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
  - Many dialect-level documents are outdated, as are the rationale ones
  - We would love help with this!

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
  - Many dialect-level documents are outdated, as are the rationale ones
  - We would love help with this!
- **Start formulating the Tensor Compiler-spanning, cross-dialect charter**
  - New documents laying out intended coherent usage across dialects

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
  - Many dialect-level documents are outdated, as are the rationale ones
  - We would love help with this!
- **Start formulating the Tensor Compiler-spanning, cross-dialect charter**
  - New documents laying out intended coherent usage across dialects
- **Start codifying flows through the Tensor Compiler**
  - E.g., integration tests spanning just the Tensor Compiler dialects
  - In addition to explicitly described flows in the charter

Tensor Compiler Design Group – **get in touch!**

Alex Zinenko
(@ftynse)

Renato Golin
(@rengolin)

Jacques Pienaar
(@jpienaar)

Matthias Springer
(@matthias-springer)

Quinn Dawkins
(@qed)

Javed Absar
(@javedabsar)

Jakub Kuderski
(@kuhar)

Suraj Sudhir
(@sjarus)

Rolf Morel
(@rolfmorel)

Andrzej Warzynski
(@banach-space)

Diego Caballero
(@dcaballe)

Kunwar Grover
(@groverkss)

Developers' Meeting

BERLIN 2025

Google

Accurate Runtime Performance Estimation for Predictably Training ML Guided Register Eviction Policies

Aiden Grossman

EuroLLVM 2025

Why do this in the first place?

Rewards are critical for training learned heuristics.

Trace-based Cost Modeling

Agenda

01 Trace Based Cost Modeling for Register Allocation

02 (Distributed) Training of Models

03 Further Improvements

Traces vs PGO Data

PGO-Based

```
.loop:
        add     eax, dword ptr [rdi + 4*rdx]
        inc     rdx
        cmp     rcx, rdx
        jne     .loop
```

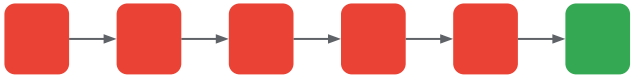
X5

```
.exit:
        add     eax, 5
        ret
```

X1

- Reward is a linear combination of instruction types multiplied by their block frequency.

Trace-Based



- Reward is some operation (typically cycles) over the sequence of instructions. We have several options to choose from:
  - Analytical CPU pipeline models.
  - ML based CPU models.
  - Raw Instruction Counting.

Sourcing Trace Data

- Previous work, llvm-mcad (EuroLLVM 2022 mshockwave@), (https://youtu.be/ZGEP7JEIKNo) that guided us down this direction used a QEMU plugin.
- Use DynamoRIO to produce traces due to good internal support.

Basic Block Trace Modeling

We need to be able to collect a single trace and apply it to many variants of the same binary as rerunning each time defeats the point.

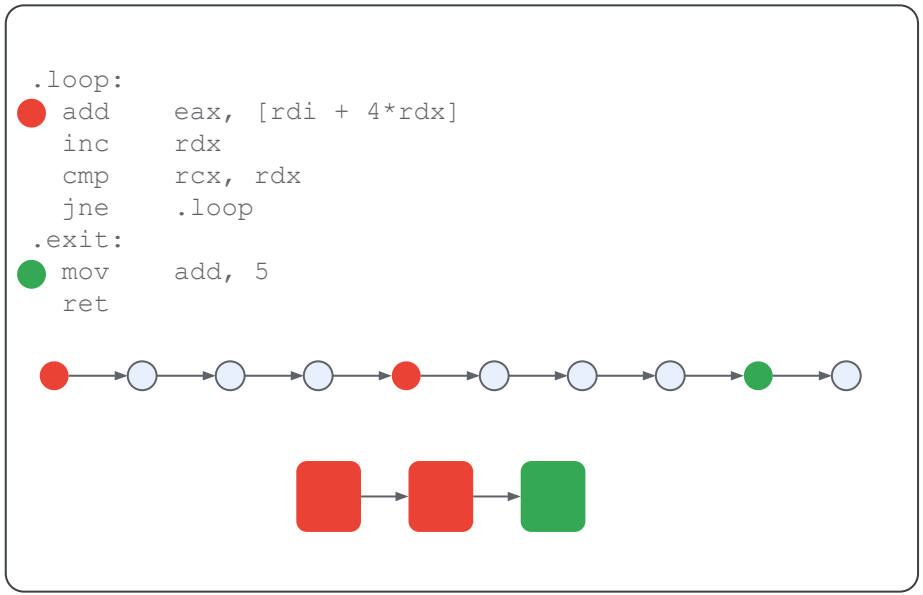
Sourcing Basic Block Information

Take Advantage of some of the Propeller Infrastructure (Basic Block Address Maps).

BB Trace Extraction

How to turn an instruction stream into a BB stream that can be replayed.

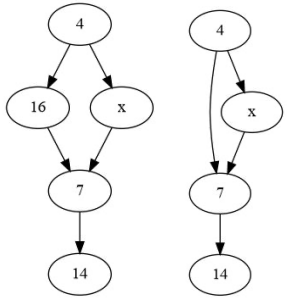
- Add a basic basic block to the trace every time we encounter an instruction with a PC starting at the beginning of a BB.
- Sometimes we need to split BBs.
  - Call Instructions
  - TCMalloc RSeq
  - Inline Assembly

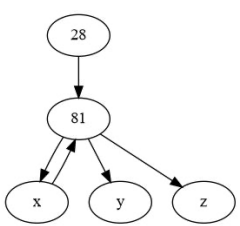


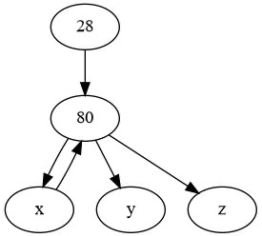
Basic Block Trace Modelling

Becomes as simple as loading BBs from the binary/compiled corpus.

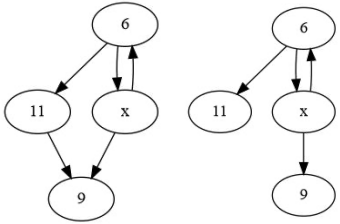
Reconciling CFG Differences - The Problem

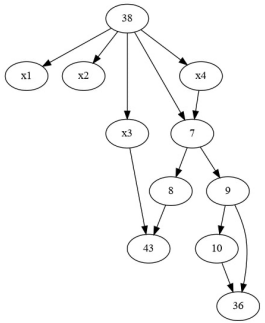


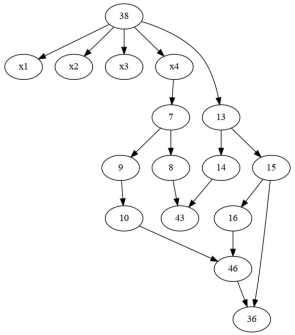




- Most cases are trivially reconcilable.
- Some cases are impossible to reconcile without additional information.







Reconciling CFG Differences - The Side Step

Disabling 3.5 Passes eliminates all CFG Differences

Findings

1. Disabling three (believed to be) non-regalloc-coupled passes eliminates all CFG differences.
2. Disable one option on another pass (the half a pass).
3. Simple trick allows for much simpler BB trace modelling design.

BB Traces are Reasonably Close to Source Traces

0.5% missing instructions

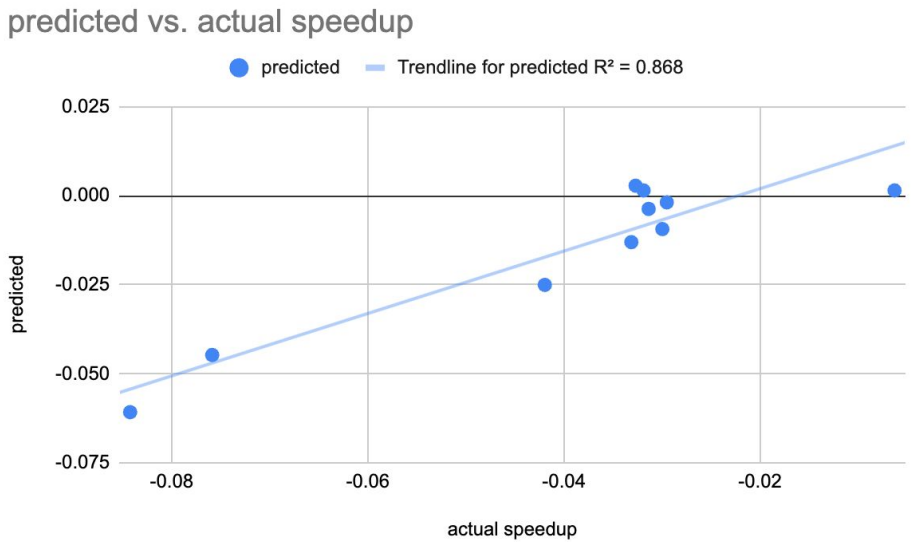
On traces upwards of 10M instructions.

Some cases we are not handling currently like interrupted restartable sequences and symbols from assembly files. No theoretical obstacles to completely fixing the gap.

It Even Works With PGO+CSPGO+ThinLTO!

For skylake, measuring runtime in cycles. Albeit with a large constant offset.

Predicted Speedup vs. Actual Speedup

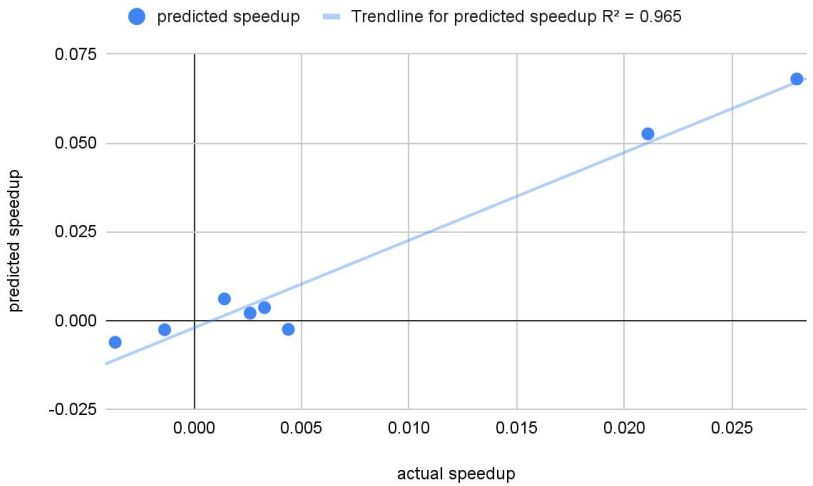


Results from LLVM opt.

The Non-PGO case works as well:

`opt -passes="default<O3>" -disable-output` on StructuralHash.cpp from LLVM. ~10M retired instructions.

Predicted Speedup vs. Actual Speedup



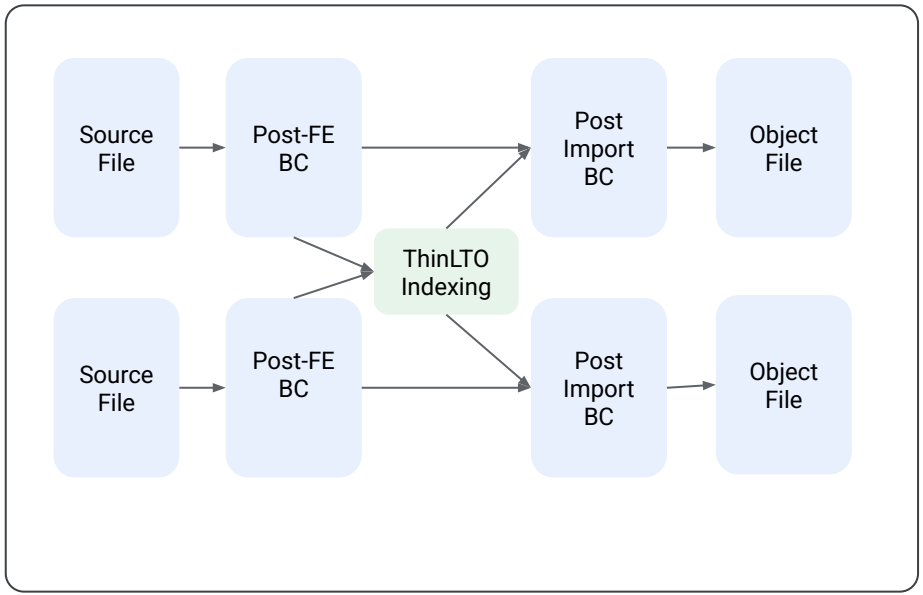
Results from LLVM opt.

Training with Reinforcement Learning

Background - The Corpus

How should we efficiently collect training data?

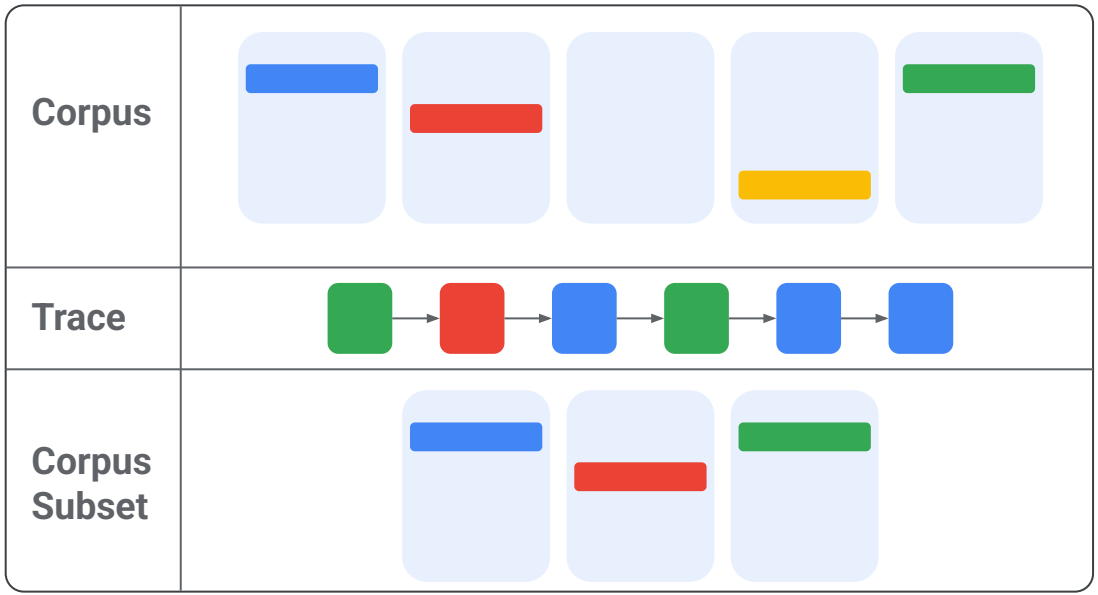
We collect LLVM Bitcode for all translation units involved in the final link.



Corpus Subsetting

How do we efficiently evaluate models?

- Find the minimum set of translation units covering the entire set of functions in the trace.
- Pull them to the side.



Background - Training Setup

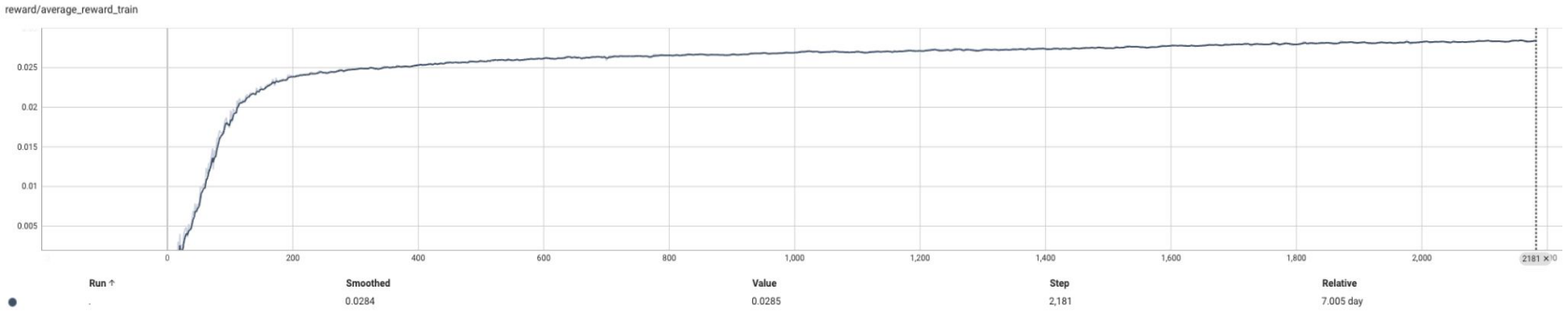
How do the requirements on the ML side interface with the cost model/compiler side?

- We use ES (Evolutionary Strategies) as our training algorithm.
  - Simple math, relatively easy to understand.
  - Enables long trajectories - We can give feedback on many individual decisions and the algorithm will still adjust the policies appropriately.
  - Has a set of perturbations for each iteration.
- Utilize existing training infrastructure
  - But scaled given now we need to compile an entire corpus subset to get a signal rather than a handful of modules.
- First experiments were performed with the same opt invocation from earlier. ~10M retired instructions.

Training Results

It trains! Somewhat slowly… (for LLVM opt)

Reward (Predicted speedup over baseline)



- 100 Machine Slices (1600 Threads)
- 100 Perturbations per iteration
- ~800 Modules
- ~7 Days of training time
- 0.5% Real World Performance Improvement over an already peak optimized (PGO+CSPGO+ThinLTO) binary.

So Why Does This Matter?

We have a validated, predictive cost model for real applications.

Distributed Training

Distributing the Training Process

Using more machines will at least help.

01

Parallelize individual workers

We use a threadpool within each worker to enable parallel compilation with the modelling component already being parallelized.

02

Spawn a bunch of distributed workers.

We use XM to manage experiments, with each worker being given about 32CPU cores, giving us reasonable scalability.

03

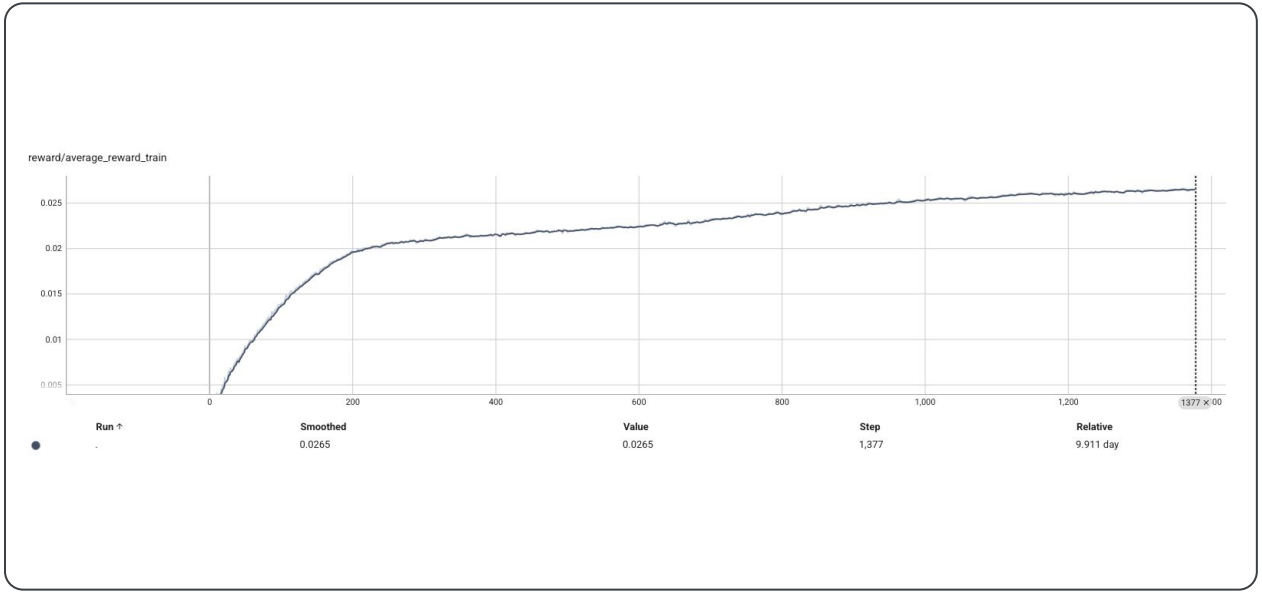
Profit (Somewhat)

This dropped iterations times to about five minutes. The latency of an individual model evaluation precludes us from going faster. We can evaluate many perturbations in parallel.

Shipping a Model

Training - Training Hyperparameter Tuning/RL

RL training started to go significantly faster when we realized the learning rate could be increased 10x with no ill-effect. More experimentation is still needed.



Some Reasonable Initial Performance Improvements

This is from a new model trained on a single workload. It ends up generalizing reasonably well.

Internal Server App 1

- 0.34ms latency on action 1
- 0.24ms latency on action 2

Internal Search App 1

```
Benchmark Setup 1:
---- total:qps ----
diff: +0.32% ±0.088%
---- <workload1 cpu kcycles> ----
diff: -0.75% ±0.198%
Benchmark Setup 2:
---- <workload2 cpu kcycles> ----
diff: -0.46% ±0.268%
```

Internal Search App 2

```
-- <round trip latency> --
diff: -0.64% ±0.372%
```

Current and Future Work

During training, only compiling functions with regalloc decisions causes compile time drops from 3-4 minutes to 5-10s.

Modeling time remains about the same as it is bottlenecked by MCA. We have some ideas on how to fix that...

Reducing Modeling Costs

Now that compile times have been drastically reduced, modeling costs dominate. It would be good to reduce them too.

- Only modeling changing functions might provide significant benefits.
  - Benefits depends upon how many functions they call.
  - Needs empirical validation.
  - Natural extensions (like excluding blocks from functions that get called) also need separate validation.
- Trace subsetting - Only evaluate a subset of the traces on each invocation.

Future Work

Future Work

Understanding Constant Offsets

- Understand why our model is producing large constant offsets.
- Hopefully leads to better models.
- Experiment with other modelling techniques (ML based, etc.)

Ship Better Models

- Utilize more efficient training techniques, train better models and ship them.

What do we need to do to Generalize for other Optimizations?

- Control flow graph reconciliation?
- Keep track of data/inputs to interpret (M)IR?
- Something else?
- Better cost modelling techniques?

Thank You!

Google

Developers' Meeting

BERLIN 2025

Google

An Update on LLVM Premerge Testing

Core Developer Software Foundations

aidengrossman@google.com

April 2025

The New Beta System

Beta Premerge System Status

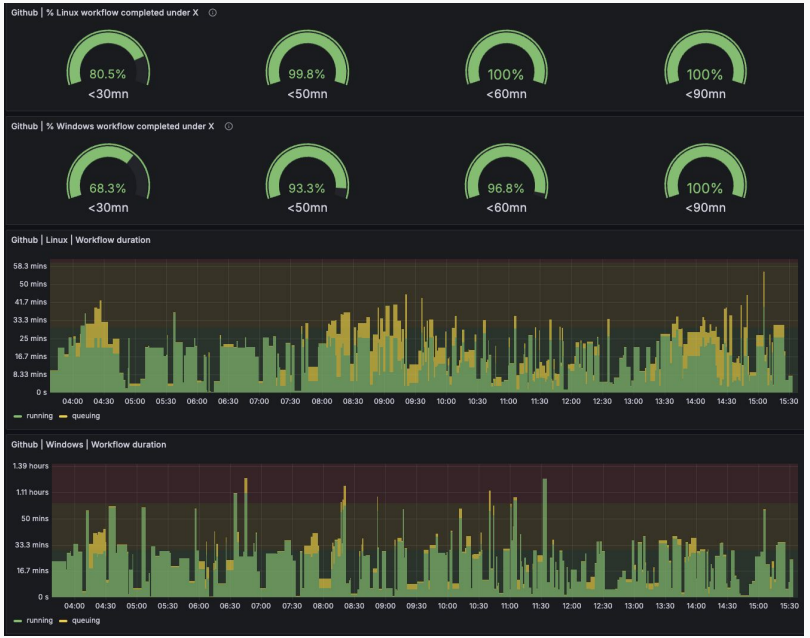
- Both old and new systems are running in parallel
- But the new system marks all jobs as passing
- Sorting out a few stability problems
- Launch expected in Q2

Beta System Features

- Autoscaling means more machines at peak load (workday afternoons Pacific time), meaning better latency and typically <20 minutes of queue time.
- Analytics
- Oncall team support during working hours
- Dedicated engineering staff for maintenance and improvements

Metrics

- Displays key metrics that have significantly impacted premerge functionality in the past.
- Metrics is coupled with alerting to notify an on-call rotation when things go awry.



Beta System Features/Improvements

Done:

- Computing what projects to test is now much simpler and unit tested.

Exploring:

- Improving node/container setup times.
- Full reproduction instructions using containers.
- Faster toolchain on Windows?

Beta System Launch Q2

- Launch in this case means marking the new system premerge tests as authoritative, meaning a failing test will report as failed to GitHub
- At this time we will also turn down the old presubmit infrastructure

Testing Resources Prioritization

- There’s always a trade-off between test coverage and premerge latency
- More test coverage means more premerge latency
- Google will do its best to deliver a high-reliability and high-performance system
- But beyond that, these tradeoffs are best made by the community
- We will defer coverage decisions to the Infrastructure Area Leads

Credits

From within Google:

- Aiden Grossman
- Caroline Tice
- Guillaume Chatelet
- Lucile Rose Nihlen
- Nathan Gauër

And the following OSS contributors:

- David Spickett
- Tom Stellard

Questions?

Thank You!

(Questions?)

Google

Developers' Meeting

BERLIN 2025

Measuring the Health of the LLVM Community

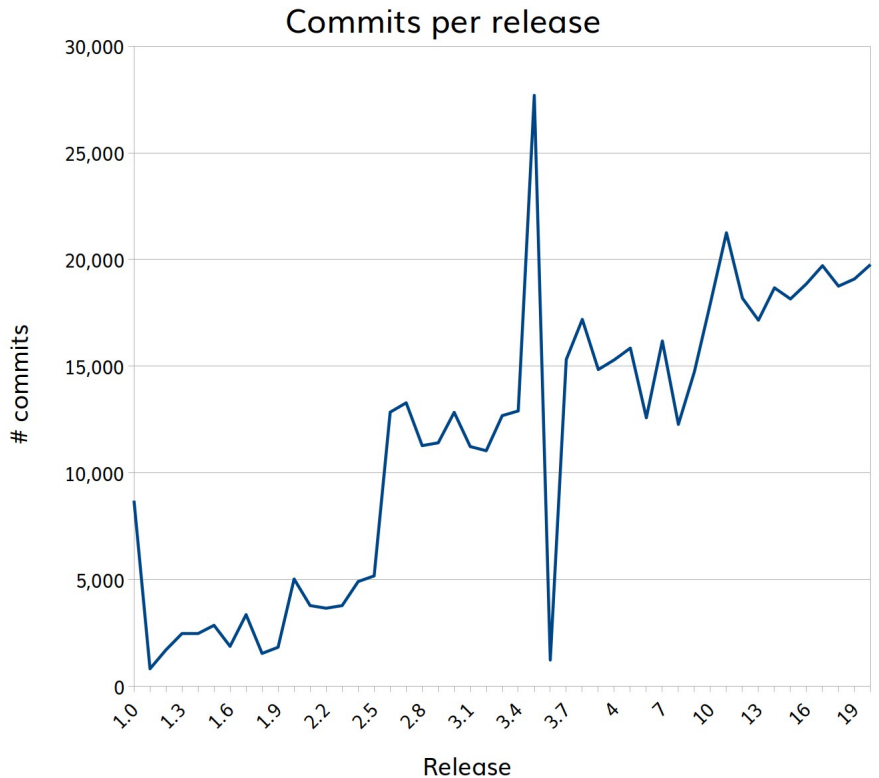
Jeremy Bennett

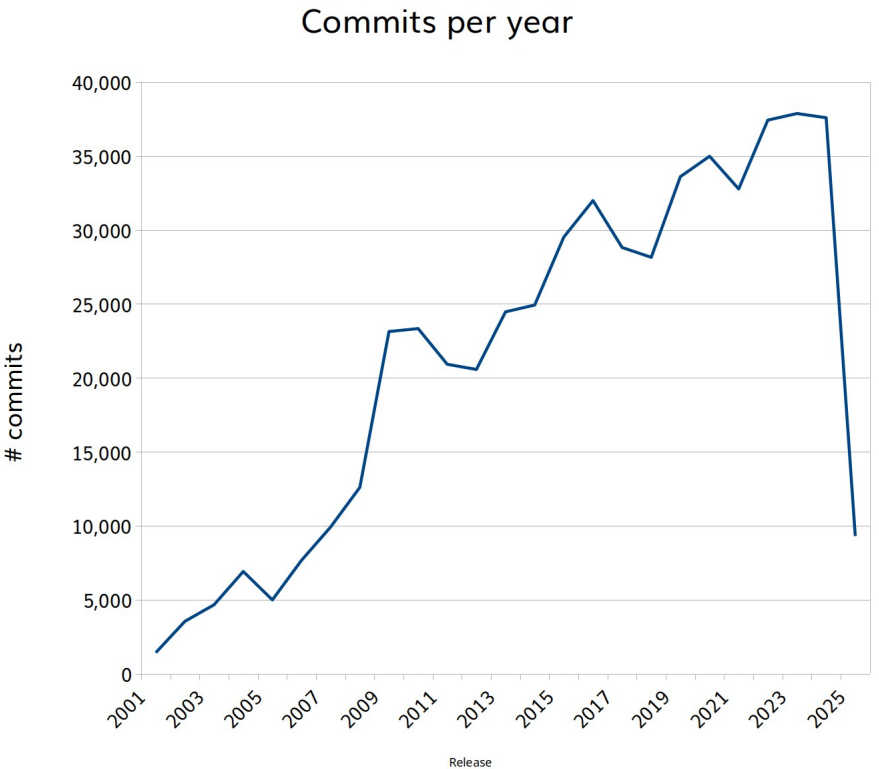
The *git* Repository

Counting commits

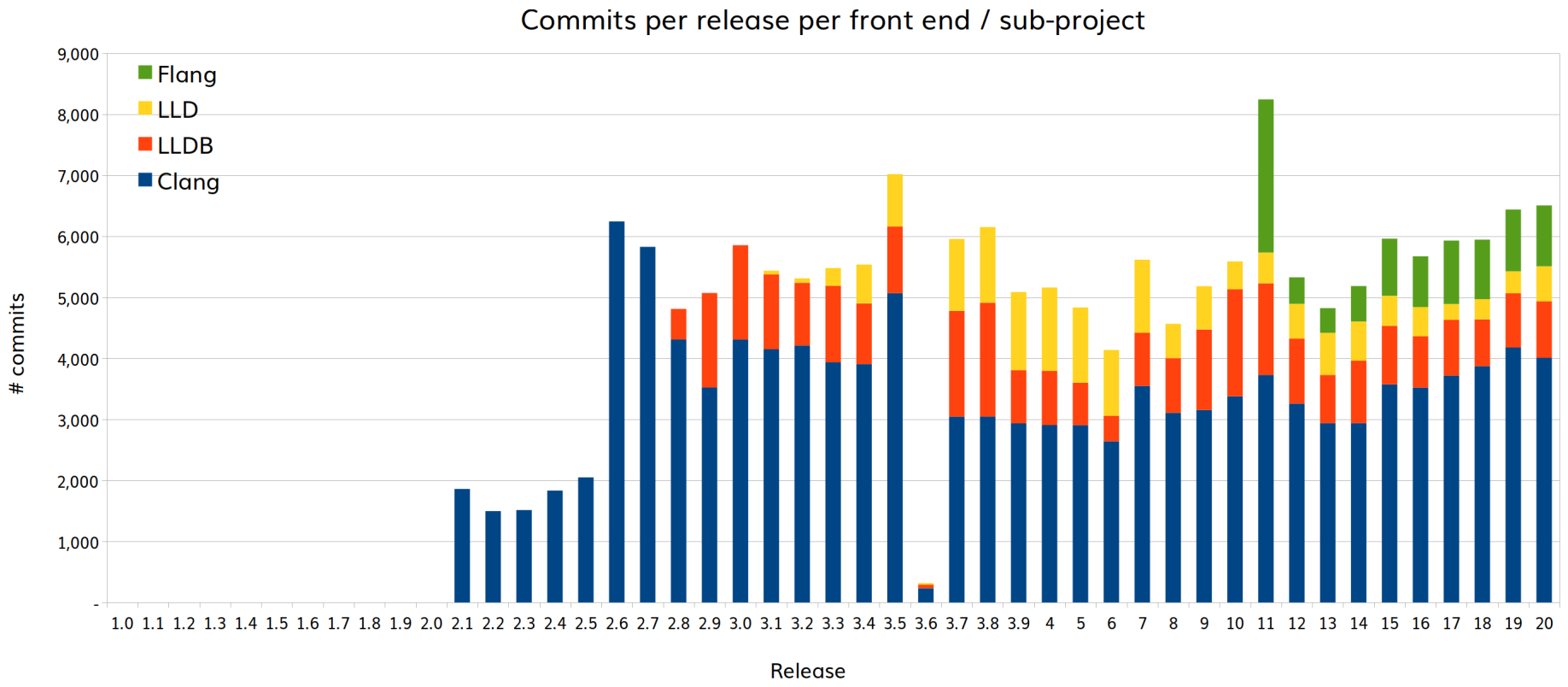
```
git log --oneline --no-merges remotes/upstream/release/20.x \
        ^remotes/upstream/release/19.x | wc -l

git log --oneline --no-merges --since-as-filter="2024-01-01" \
        --until="2024-12-31" | wc -l
```

Counting Commits

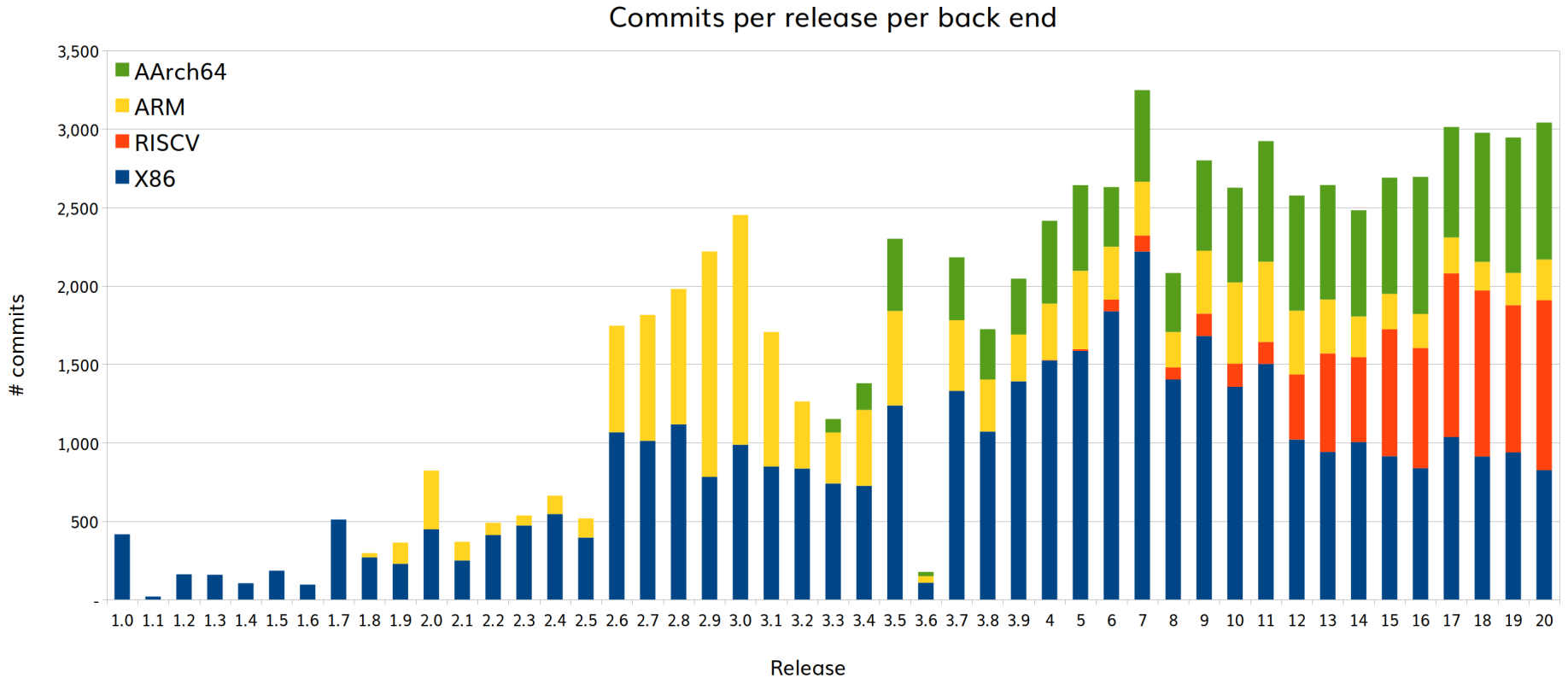




Front End & Sub-Project Activity (Commits)



Back End Activity (Commits)

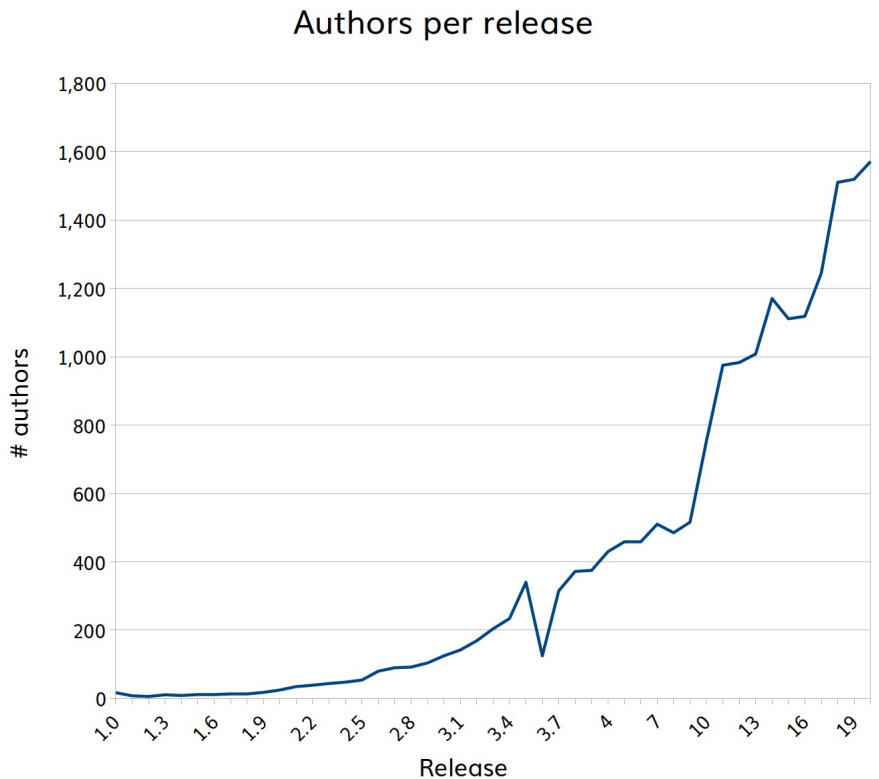


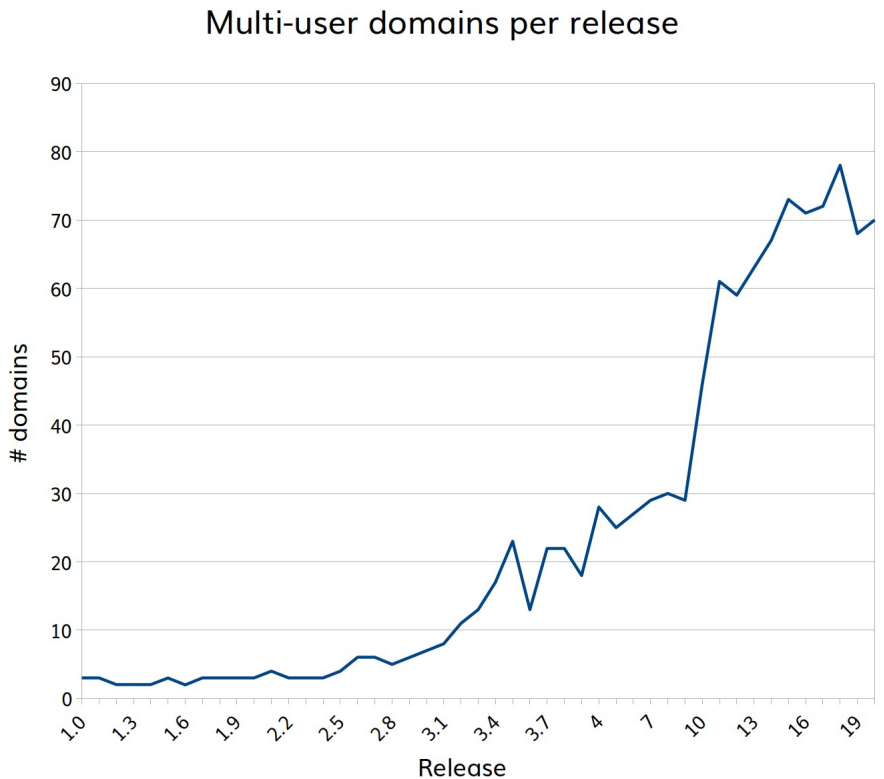
Counting Contributors

```
git log --no-merges remotes/upstream/release/20.x \
        ^remotes/upstream/release/19.x \
        --pretty="format:%an" | sort -u | wc -l

git log --no-merges remotes/upstream/release/20.x \
        ^remotes/upstream/release/19.x \
        --pretty="format:%ae" | sed -e 's/^.\+@//' | sort -u
```

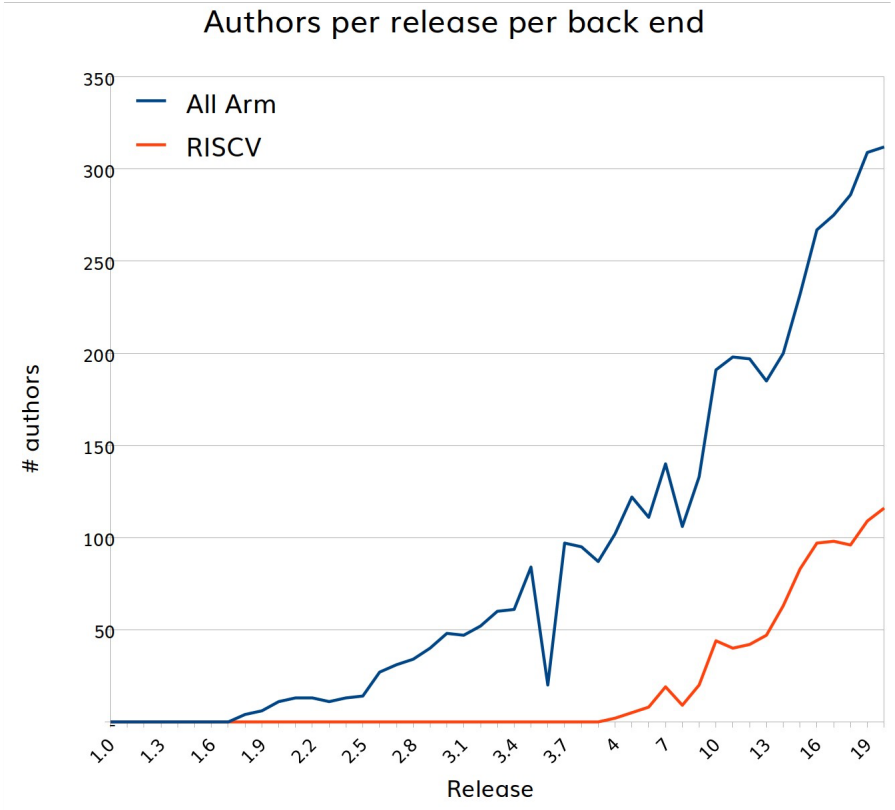
Then post-process to remove generic email domains (gmail etc) and any domain with a single contributors

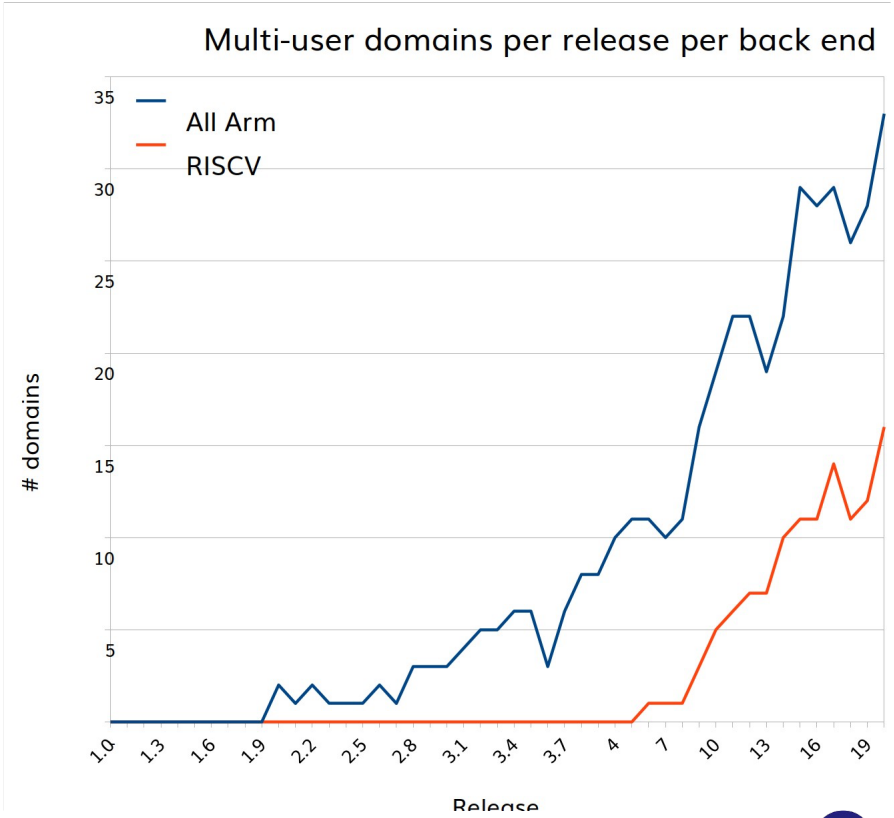
Counting Contributors





The Detail: Comparing Back Ends





The Detail: LLVM Supercommitters

Release 20

- 1,571 individuals
- 70 “corporate” domains
- 63 individuals committed more than once per week
- Highest individual: 767 commits

The Detail: LLVM Supercommitters

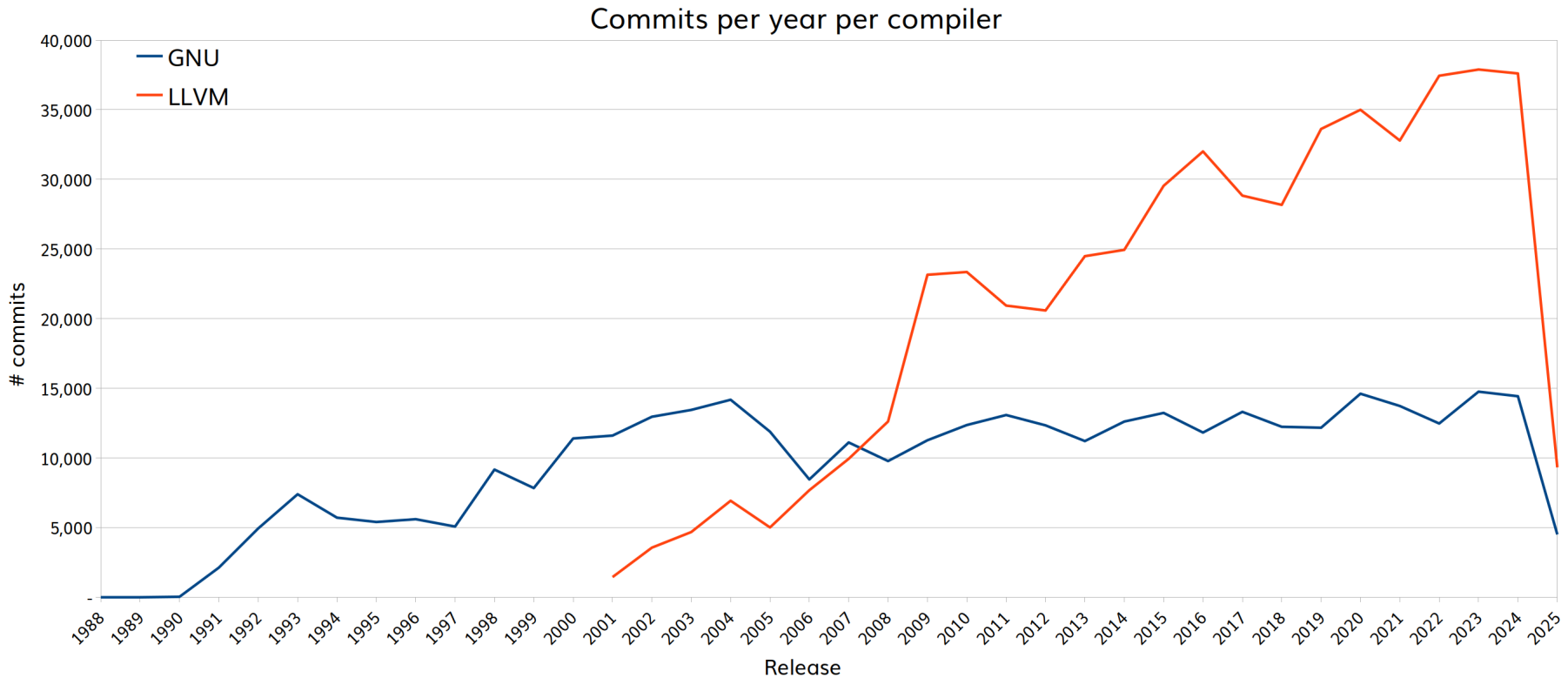
Release 20

- 1,571 individuals
- 70 “corporate” domains
- 63 individuals committed more than once per week
- Highest individual: 767 commits

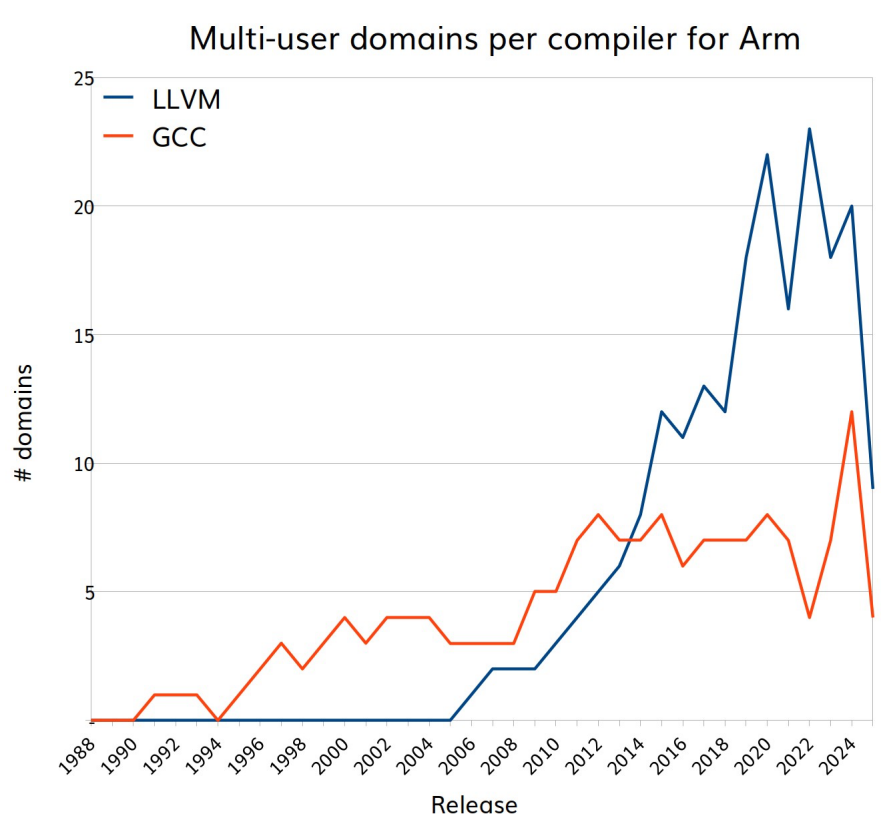
Release 1.0

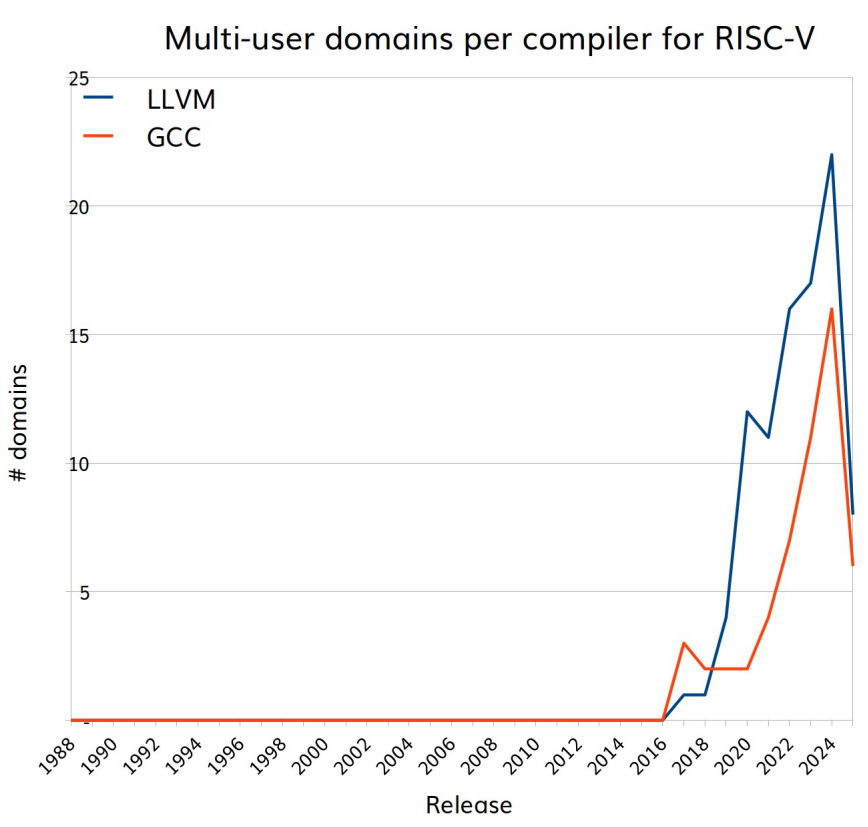
- 17 individuals
- 2 “corporate” domains
- 7 individuals committed more than once per week
- Highest individual: 6,511 commits

Comparing Compilers: Commits per Year



The Detail: Comparing Compiler Back Ends





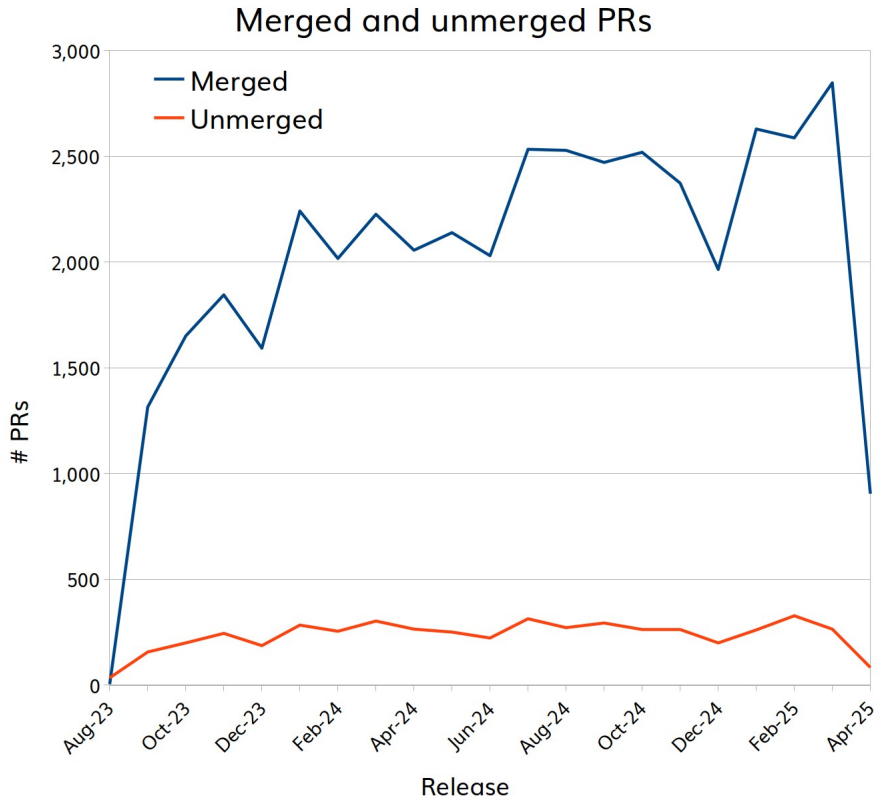
GitHub using *gh*

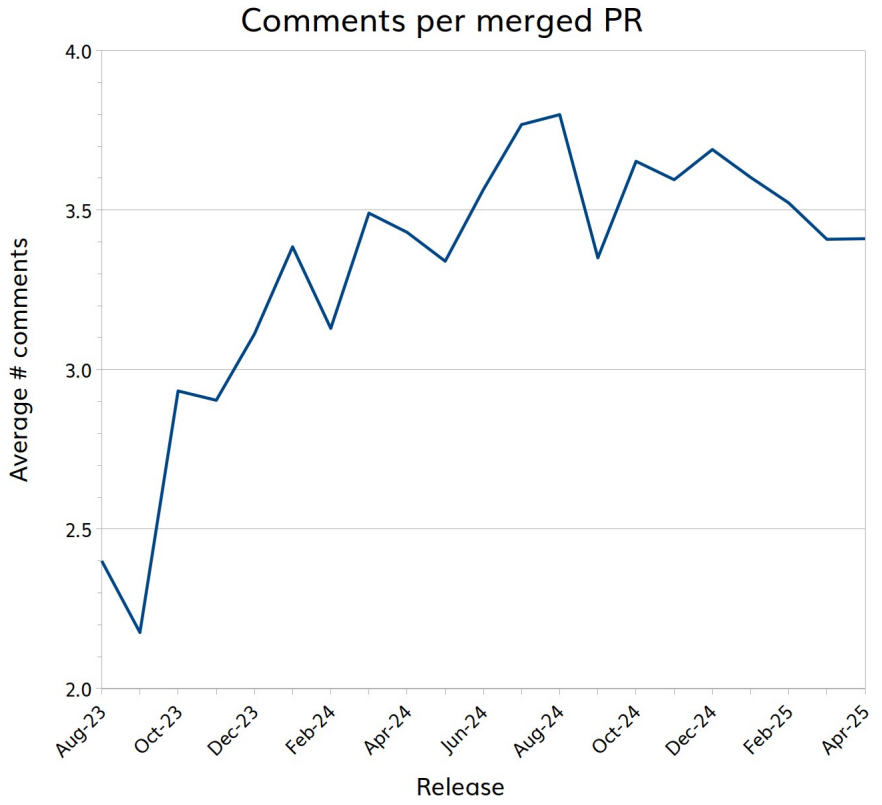
Analysing Pull Requests

```
gh pr list -L 1000000 -s closed \
           --json id,createdAt,closedAt,state,comments \
           -q '.. | .id?,.createdAt?,.closedAt?,.state?'
```

Post process with an *awk* script to generate a CSV file.

PRs: Comments per Commit





EMBECOSM®

More Detailed Analysis: Diversity and Inclusion

The Inspiration

Prof Andrea Capiluppi
University of Groningen

- www.rug.nl/staff/a.capiluppi/
- Infer gender activity
  - how robust is this?
- Statistical analysis
- youtu.be/sTAtuSxwCss
- doi.org/10.1093/iwc/iwt047

Thank You

jeremy.bennett@embecosm.com

github.com/embecosm/toolchain-analyze

embecosm.com

Jeremy Bennett

Developers' Meeting

BERLIN 2025

Lessons learned from leveling up RISC-V LLVM testing

Alex Bradbury asb@igalia.com

EuroLLVM, 2025-04-15

Background

- RISC-V
- LLVM's buildbot infrastructure
  - Important: this is focused on post-commit CI

Challenges

Challenge:

Lack of high performance commodity RISC-V hardware with support for all desired instruction set extensions.

Challenge:

A full bootstrap build and test under qemu-system is incredibly slow (>16h).

Challenge:

Limitations in bug finding ability of a two-stage Clang build followed by running LLVM's unit tests.

Challenge:

Missing a flow for iterative developing and testing builder configurations prior to commit and deployment from llvm-zorg.

Challenge:

Existing buildbot worker configurations/recipes are hard to customise to the needs of the RISC-V setup. e.g. cross-compilation followed by running tests under qemu-system.

Challenge:

Difficult to reproduce a failure locally.

Challenge:

It can be time consuming to manually root cause a regression with multiple candidate commits.

Challenge:

There is no single “best” builder approach given the pros/cons of different emulation choices and trade-offs of test coverage and speed.

Challenge:

The buildbot interface is hard to navigate to check the status of RISC-V bots at a glance, so failures can go unnoticed.

Dashboard: https://igalia.github.io/riscv-llvm-ci/

RISC-V LLVM CI status

Generated at 2025-04-06 13:59. All times were recalculated in your local timezone (Europe/London). Regenerated approximately every 20 minutes.

Bot	In progress build	Previous build
clang-riscv-rva20-2stage ▸ Info	#802 1h31m ago	#801 1h32m · 2025-04-06 12:28
clang-riscv-rva23-evl-vec-2stage ▸ Info	#641 13m ago	#640 1h57m · 2025-04-06 13:46
clang-riscv-rva23-2stage ▸ Info	#531 5h59m ago	#530 15h21m · 2025-04-06 07:59
libc-riscv64-debian-dbg ▸ Info		#12799 9m · 2025-04-06 12:36
libc-riscv64-debian-fullbuild-dbg ▸ Info	#11903 1m ago	#11902 4m · 2025-04-06 12:41
libc-riscv32-qemu-yocto-fullbuild-dbg ▸ Info	#6722 1m ago	#6721 37m · 2025-04-06 13:36

This dashboard and the `clang-*` bots operated by Igalia, supported by RISE.

Thank you

- Patch reviewers
- Everyone contributing to development and upkeep of LLVM's testing infrastructure.
- RISE for supporting this work.

Questions?

asb@igalia.com

Overflow

- Release testing

Future work:

- Performance testing and tracking.
- Wider ecosystem testing (e.g. large corpus of Linux packages)
- On-demand pre-commit CI.

Developers' Meeting

BERLIN 2025