

# SIG OpenXLA Community Meeting

---

December 13, 2022

# What is OpenXLA?

---

**Open, state-of-art ML compiler ecosystem, built collaboratively with Hardware & Software partners, using the best of XLA & MLIR.**

# Introductions

---

# Welcome!

- Welcome to any new attendees? What are you looking to focus on?
- SIG member organizations:
  - Alibaba
  - AMD
  - Apple
  - ARM
  - AWS
  - Google
  - Intel
  - Meta
  - NVIDIA

# SIG Collaboration

---

Reference material for our new collaborators

# Our Meetings

- Monthly on Zoom, 3rd Tuesday @ 8AM PT (except this week)
- Rotating meeting host & scribe
- Proposed agenda shared by host week prior in [GitHub Discussions](#)
- Meeting minutes & slides shared publicly in the [meetings archive](#) the day after
- Meetings should include:
  - Development updates
  - Design proposals
  - Community topics

# Our Collaboration Channels

Channel	Content	Access	Archive
<a href="#">GitHub organization</a>	Code, Design proposals, PRs, Issues, Roadmaps	Public	N/A
<a href="#">Community repository</a>	Governance, Meetings, Code of conduct	Public	Public
<a href="#">Community discussions</a>	Meta discussions on openxla/community repo	Public	Public
<a href="#">Technical discussions</a>	Technical discussions on individual repos: xla, stablehlo	Public	Public
<a href="#">Discord</a>	Sync discussions	Open invites	Archived chats
<a href="#">Community meetings</a>	Monthly live meetings	Public	Public agenda, slides, meeting minutes

# Development Updates

---



# Cost analysis-based fusion in XLA:GPU

---

Ilia Sergachev, Google

[sergachev@google.com](mailto:sergachev@google.com)

# Agenda

- Motivation
- HLO cost analysis
- GPU performance modeling
- Fusion based on analysis and modeling
- Results

**We **greatly encourage** community feedback in the Q&A and in our GitHub and Discord channels**

# GPU performance overview

- Compute: ~100 TB/s, DRAM access: ~1 TB/s
- ~100'000 threads; best use: doing same operation; no inter-thread communication
  - Next-best: communication at small group (~32-1024 threads) level
  - Better not: communication between cores through DRAM

Therefore:

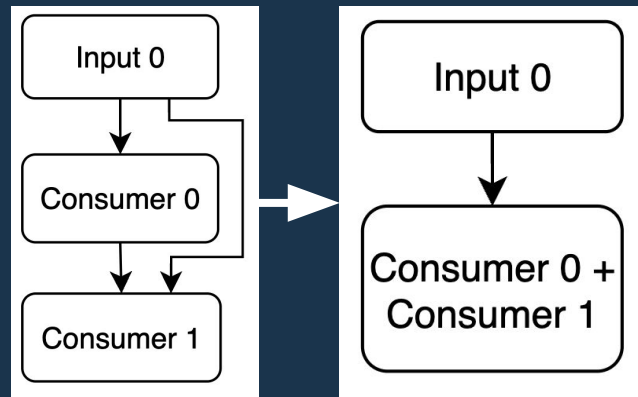
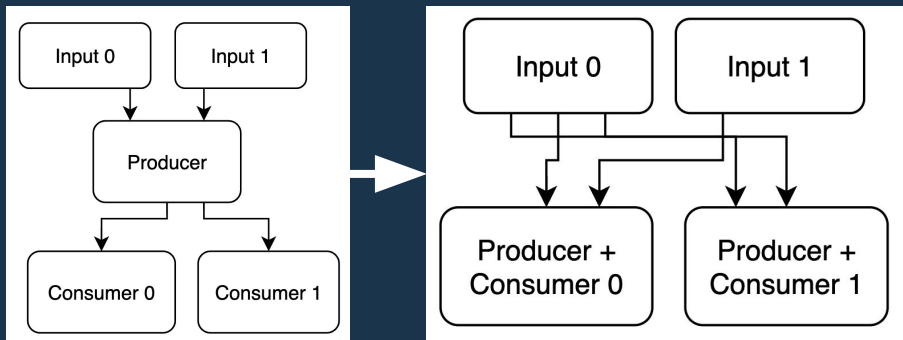
- -> Split problems into parallel independently computable pieces
- -> Allow some operations to be recomputed by multiple threads to save on communication
  - **But how much recomputation is OK?**

# Compilation process


- Input: HLO graph
- Optimizations
- **Fusion**
- LLVM IR generation (emission)
- Buffer assignment, scheduling, ...

# Fusion

- Avoiding materialization of intermediate results in DRAM
  - Sometimes by replicating parts of HLO graph: trading repeated computation for less memory access
- Avoiding repeated access to common inputs



# Fusion pipeline

- 
- Instruction fusion: trivial checks
    - Merges producers -> consumers
    - Starts new fusions where growing existing ones is non-trivial
  - Fusion merger + multi-output fusion
    - Considering fusions, not single instructions -> can afford deeper analysis
    - **Decision at every possible merge: is the execution faster after merge?**

# Fusion merging: is the execution faster after merge?

- Estimate and compare the execution times merged vs non-merged
- Execution time = HLO costs (compute, mem access)  $\times$  GPU performance
- Account for input/output access times and compute time
- Account for kernel launch overhead  $\sim$  few  $\mu$ s
- Account for utilization of GPU resources: do we use enough threads / SMs / FPUs?
- -> Best estimates if exact target GPU parameters are taken (frequencies, hardware unit counts, cache sizes)

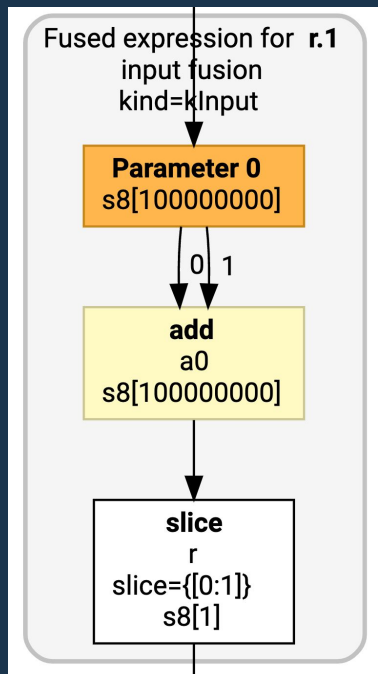
# HLO cost analysis

- Each HLO operation: input and output shapes + simple math in between
- Implementation affects cost
  - Reads input multiple times? (matrix multiplication etc)
  - Approximate or exact? (divide, sqrt, ...)
  - Uses dedicated hardware unit (data + operation type)?
  - ...
- Each input can be used **elementwise or not**
  - -> defines if it stays in one thread's registers or is recomputed by different threads



# HLO cost analysis: graph traversal reflecting emitter behavior

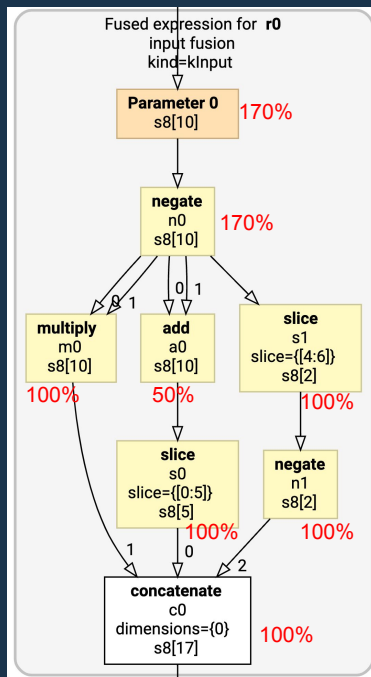
Complete fusion graph traversal is necessary because of non-elementwise operations



Only 1 byte is accessed  
despite the nominal parameter size

# HLO cost analysis: graph traversal reflecting emitter behavior

Accumulate HLO graph node utilizations: reverse post-order DFS (all users before producer)

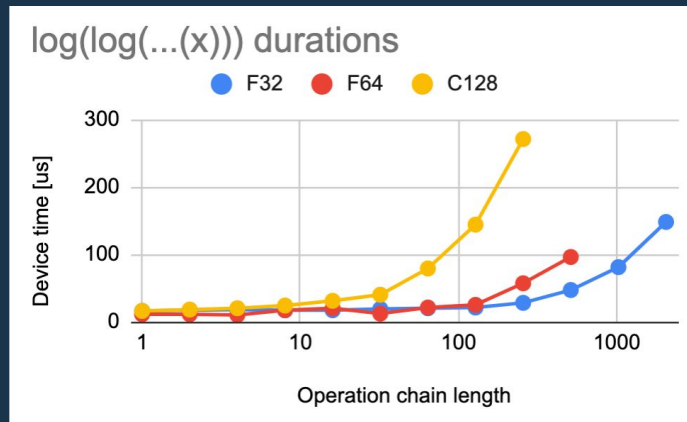


Propagate utilizations up  
from root to parameters

Root: always 100%

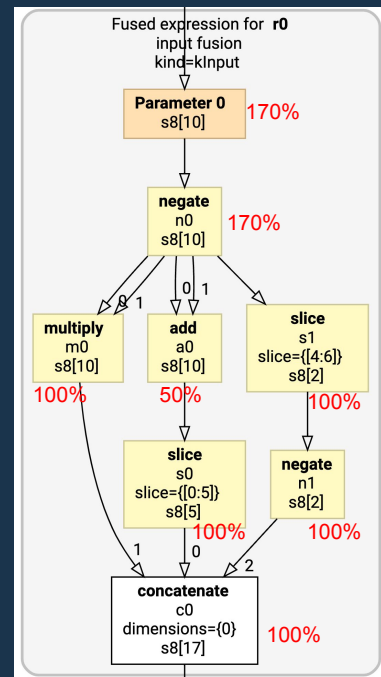
# HLO cost analysis: FLOPS

- $\log(x)$  is fast
- $\log(\log(\log(x)))$  is still fast
- ... but not that fast if  $x$  is float64
- How many operations per thread make a kernel compute-bound?
- -> for all math operations FLOPS estimates are required



# HLO cost x GPU performance -> run time

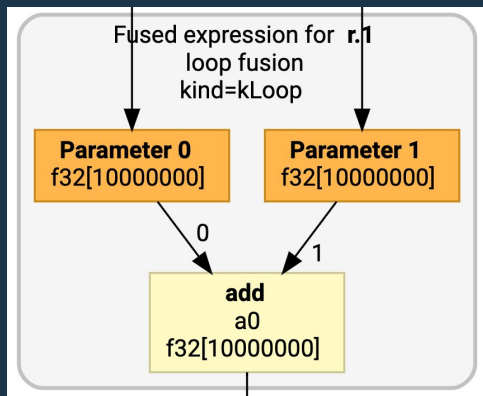
- Output write time: output size / DRAM bandwidth
- Input access time: (input size \* input utilization) / memory bandwidth
  - DRAM for large inputs; L1 / L2 caches for small ones
- Compute time:  $\Sigma(\text{FLOPS of HLO node} * \text{utilization}) / \text{GPU FLOPS/s}$ 
  - Adjusted if kernel is not using all hardware cores
- Kernel launch overhead (few  $\mu\text{s}$ )



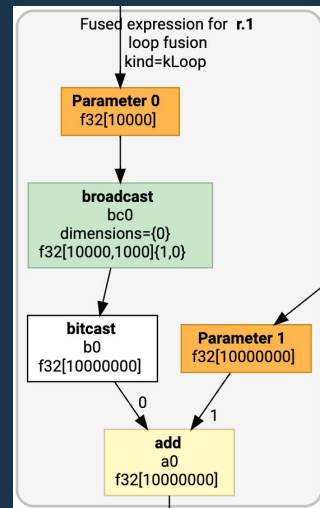
# HLO cost x GPU performance -> run time: examples

Accounting for cache effects is important:

these 2 examples have the same total amount of memory access and computation.



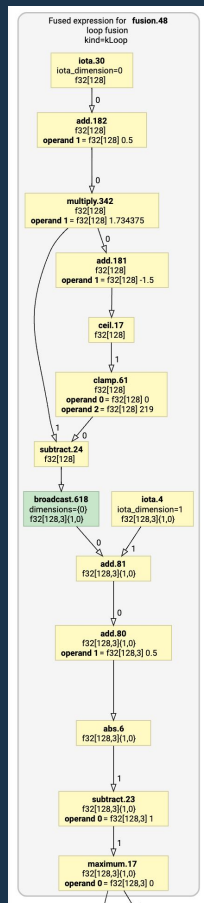
175 us



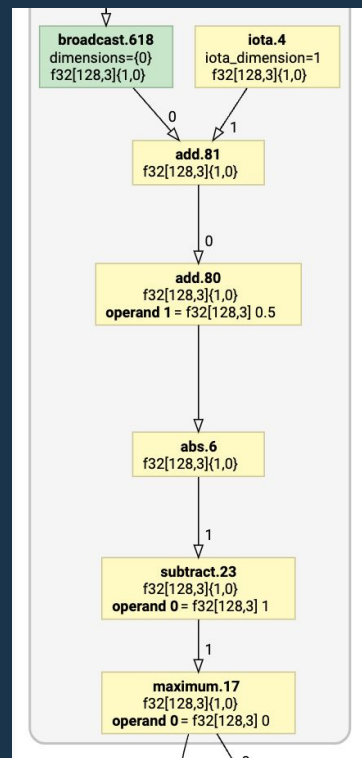
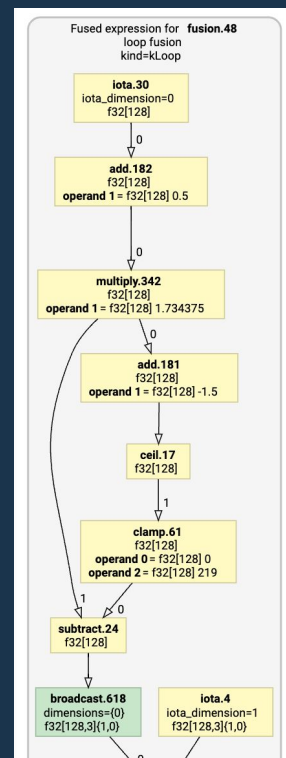
118 us

# Fusion decision taking: example

- 384 \* F32 elements,  
simple math

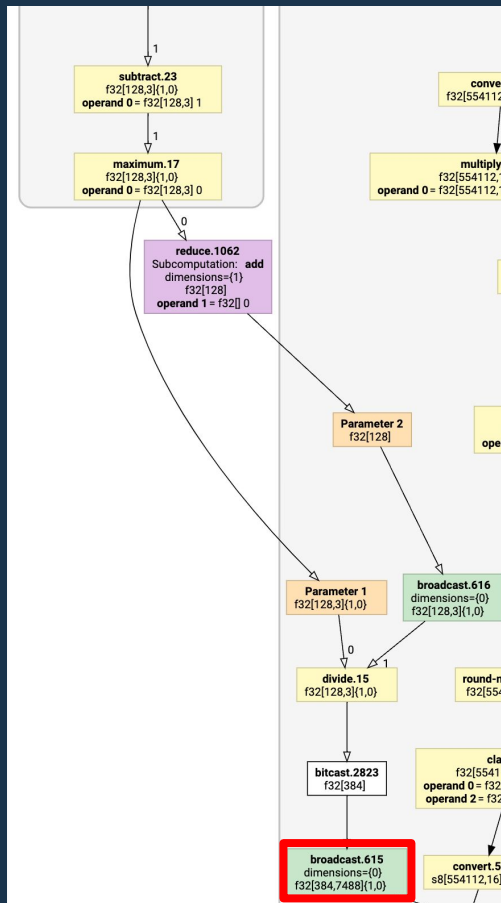
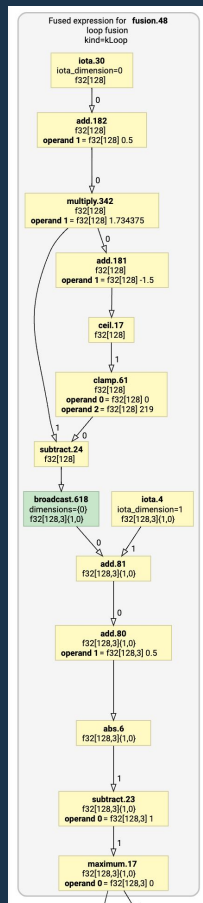


(zoom in):



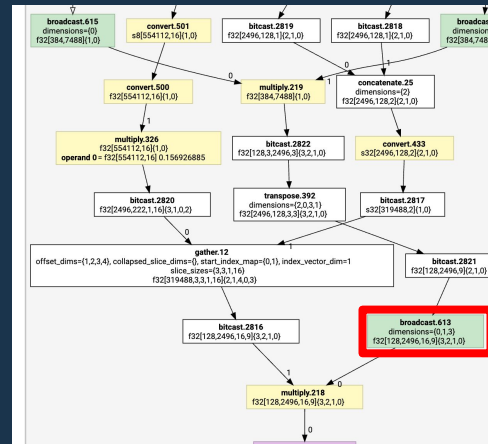
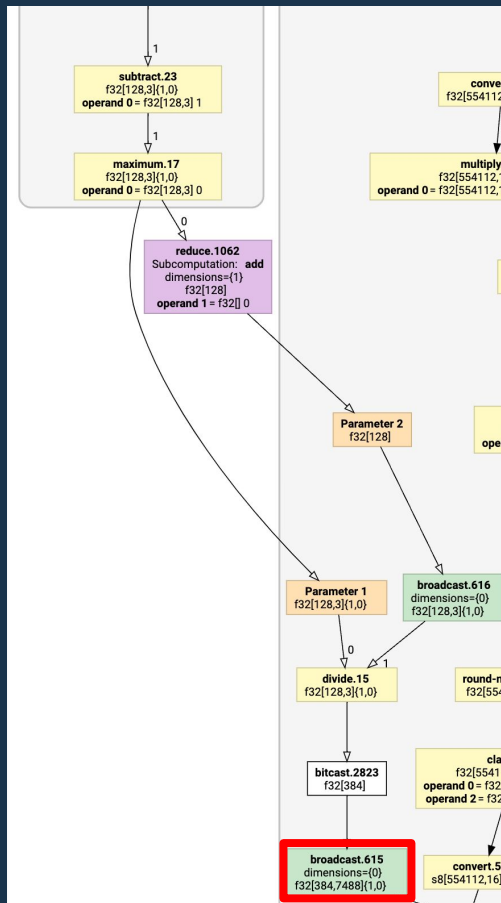
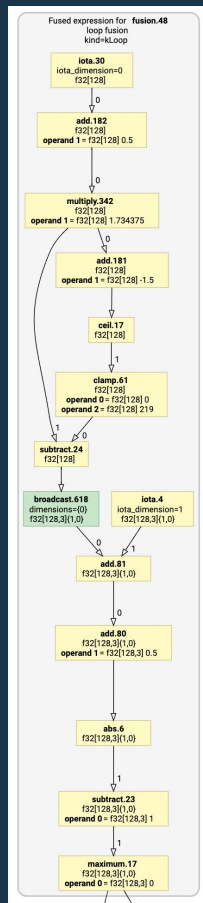
# Fusion decision taking: example

- $384 * F32$  elements, simple math
- Broadcast x7488



# Fusion decision taking: example

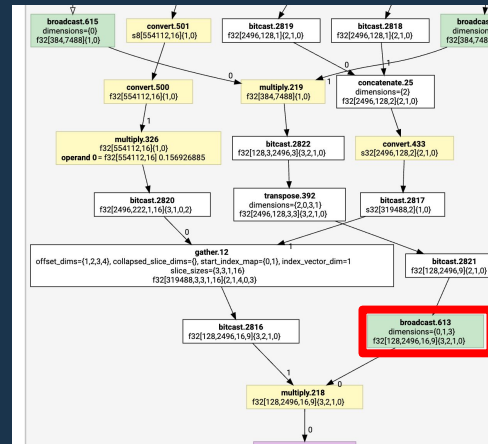
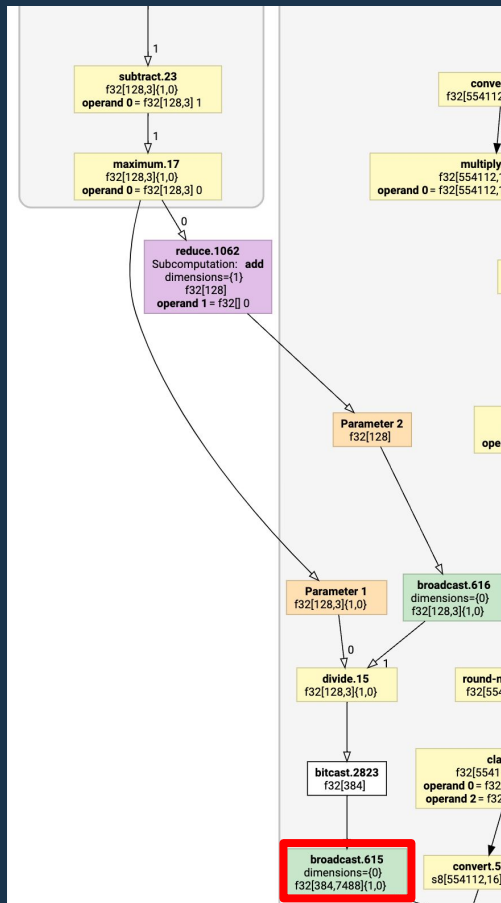
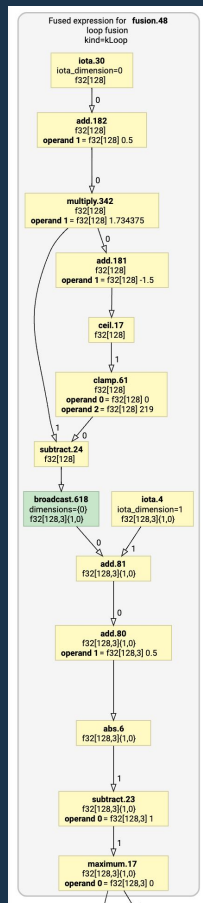
- 384 \* F32 elements, simple math
- Broadcast x7488
- Broadcast x16





# Fusion decision taking: example

- 384 \* F32 elements, simple math
  - Broadcast x7488
  - Broadcast x16
- > **Better not merge!**  
(3x as fast)



# Results

- Larger fusions
- Faster execution
  - ~30% end-to-end speedup on a T5 model by avoiding compute-boundness
  - 5-10% end-to-end speedups on a variety of benchmarks
- Faster compilation (2x on some benchmarks)  
by reducing register spilling and excessive IR size
- Better profiling because of correctly counted input access volume

## Current assumptions and constraints

- Producers are *loop* fusions (~1 thread / output element)
- Consumers have ~1 thread per input element (by design of fusion pipeline)
- GPU is good at latency hiding:  $t = \max(t_{\text{access}}, t_{\text{compute}})$
- Small inputs stay in caches
- Caches are empty at kernel launch (generally not true, but difficult to account for)
- Perfect memory coalescing (optimized in other parts of XLA pipeline)
- No vectorization (currently separate logic applied after fusion)
- Implementation details are in sync with the cost model

# Q&A

---

# OpenXLA Governance

---

Thea Lamkin, Google

[thealamkin@google.com](mailto:thealamkin@google.com)

# Goals

- Decisions can be made efficiently *and* inclusively
- Escalations are used rarely
- Governance supports increasing project complexity
- Contributors can easily find what they need to participate
- Technical leadership is diverse and rewards continued investment
- Collaboration happens constructively and respectfully

# Project References



# OpenXLA Governance

- Small groups make consensus-driven decisions
- Escalation process is clearly defined
- Structure is hierarchical and scalable
- Processes are transparent and predictable
- Governance documents are clearly written and practical
- Technical responsibility tracks with contributions
- All community members are help to common standards

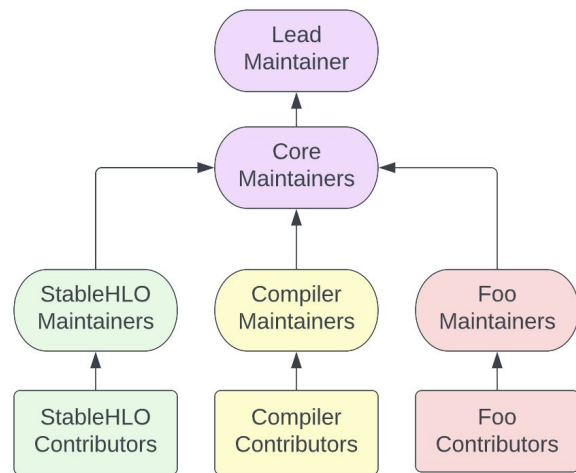


# Governance Structure

Hierarchical:

- A community of **contributors** who file issues, make pull requests & proposals, and contribute to the project.
- A small set of **module maintainers** drive each module of the OpenXLA project.
- They are supported by **core maintainers**, who drive the overall project direction.
- The core maintainers have a **lead core maintainer** who is the catch-all decision maker.

Technical governance is strictly separated from business governance. Membership in the technical governance process is for **individuals**, not companies



## Governance Structure cont.

An [Interim Steering Group](#) will (transparently) steward creation of the appropriate structures & processes to implement the governance model, including:

- Governance documentation
- Code of Conduct
- Key processes: RFCs, process
- Owners files, etc.

[Other maintainer groups](#) may be considered to support non-technical project areas:

- Business & Marketing Collaboration
- Community Management
- Documentation

However, the above groups will defer all technical decisions to Core & Module Maintainers.

# Timeline

<b>Dec 13</b>	Governance proposed @ Community Meeting
<b>Dec 13</b>	Governance RFC opened in /community
<b>Jan ?</b>	Optional deep dive on Governance RFC
<b>Jan 17</b>	Governance RFC discussed @ Community Meeting
<b>Jan 17</b>	Governance RFC accepted or revision planned
<b>Jan 31</b>	Interim Steering Group publishes governance roadmap
<b>Feb 21</b>	ISG reports progress

# How to Contribute

- Review the Governance proposal [here](#).
- Provide feedback by [January 13, 2022](#) on GitHub or by emailing [thealamkin@google.com](mailto:thealamkin@google.com)

# Q&A

---

# Update on OpenXLA and IREE

---

# Let's continue to discuss on GitHub!

[github.com/openxla/xla/discussions](https://github.com/openxla/xla/discussions)