UCI University of California, Irvine

MCA Daemon

Hybrid Throughput Analysis Beyond Basic Blocks

Min-Yih “Min” Hsu, David Gens, Michael Franz. University of California, Irvine

Keynote, EuroLLVM 2022

Outline

Outline

Motivation

Outline

Motivation

MCA Daemon (MCAD)

Outline

Motivation

MCA Daemon (MCAD)

Future Plans

Outline

Motivation

MCA Daemon (MCAD)

Future Plans

Epilogue

Outline

Motivation

MCA Daemon (MCAD)

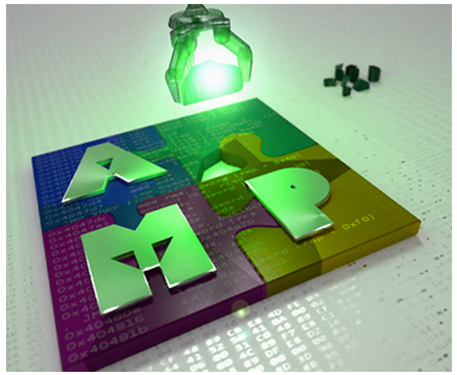
Future Plans

Epilogue

Genesis: Assured Micro Patching (AMP)

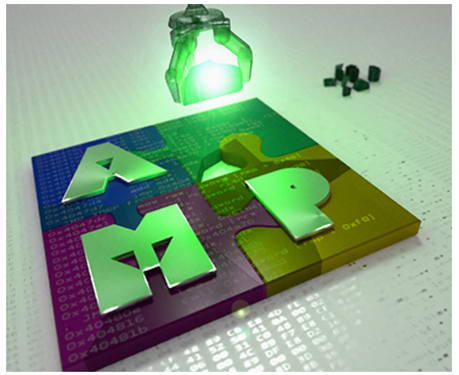
Genesis: Assured Micro Patching (AMP)

- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code



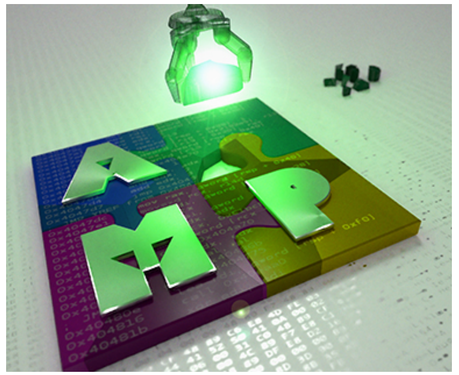
Genesis: Assured Micro Patching (AMP)

- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code
  - Including *functional* and *timing* aspects



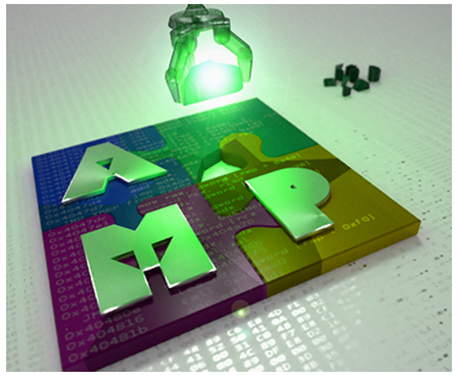
Genesis: Assured Micro Patching (AMP)

- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code
  - Including *functional* and *timing* aspects
  - Focuses on **small (micro)** binary patches



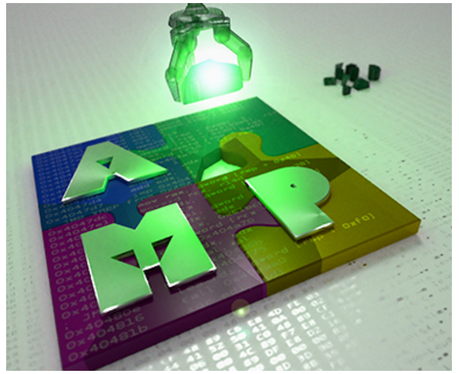
Genesis: Assured Micro Patching (AMP)

- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code
  - Including *functional* and *timing* aspects
  - Focuses on **small (micro)** binary patches
  - Focuses on **embedded systems**



Genesis: Assured Micro Patching (AMP)

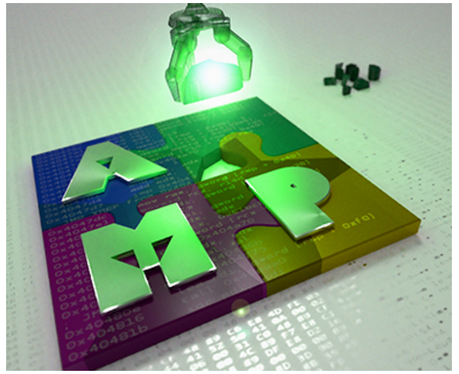
- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code
  - Including *functional* and *timing* aspects
  - Focuses on **small (micro)** binary patches
  - Focuses on **embedded systems**



- UCI was studying the timing impacts of binary patches

Genesis: Assured Micro Patching (AMP)

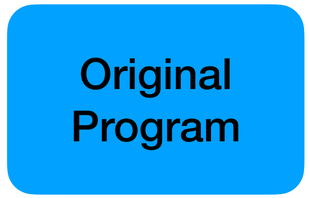
- A research project initiated by United States DARPA to assure the correctness of **binary patching** with little or *no* source code
  - Including *functional* and *timing* aspects
  - Focuses on **small (micro)** binary patches
  - Focuses on **embedded systems**



- UCI was studying the timing impacts of binary patches
  - Example: After the firmware on a truck is binary-patched to prevent brakes from locking up, we need to make sure **latencies** do not degrade terribly

Timing impacts of binary patches

Problem definition

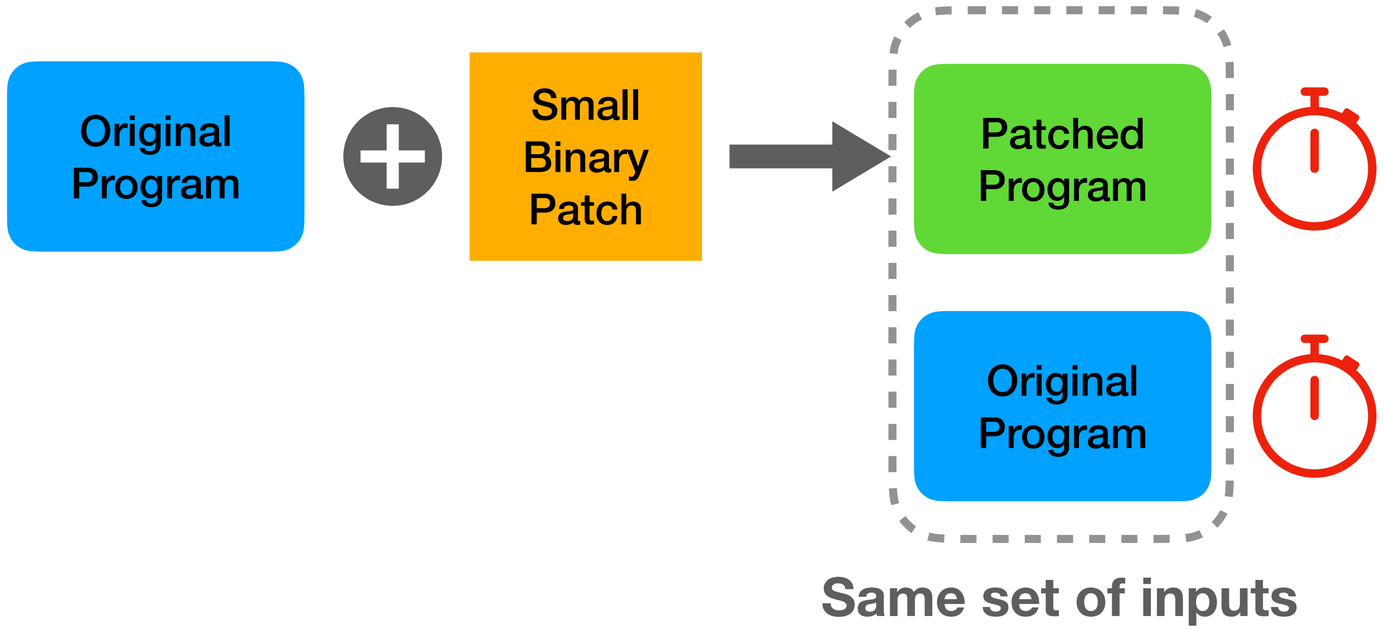


Timing impacts of binary patches

Problem definition

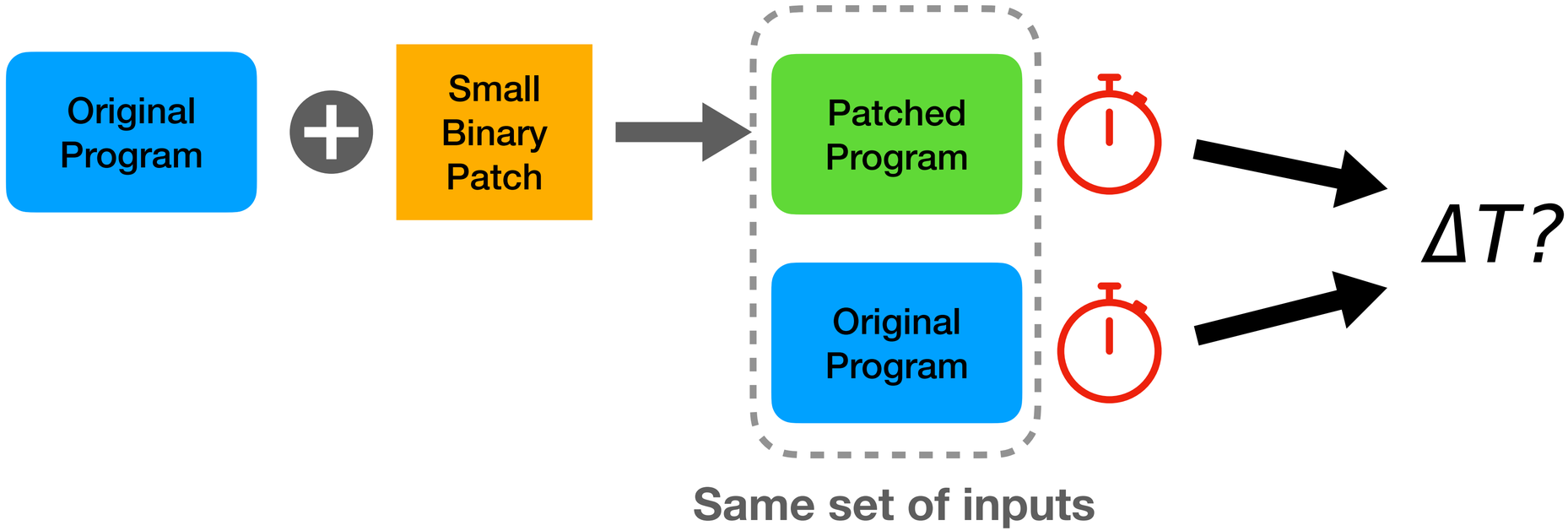
Timing impacts of binary patches

Problem definition

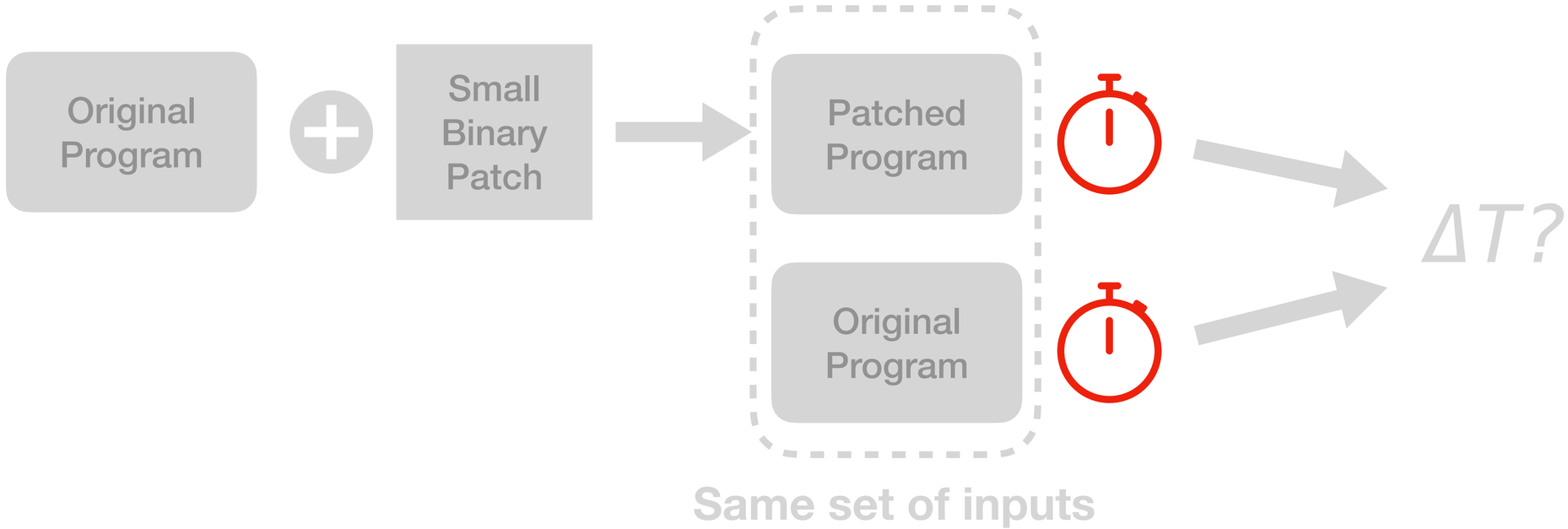


Timing impacts of binary patches

Problem definition



Execution time assessment



Execution time assessment

Interesting use cases

Execution time assessment

Interesting use cases

- Predicting program run time in **remote** environments or time-sensitive applications
  - Examples: firmware in cars or satellite (e.g. Kepler space telescope by NASA)

Execution time assessment

Interesting use cases

- Predicting program run time in **remote** environments or time-sensitive applications
  - Examples: firmware in cars or satellite (e.g. Kepler space telescope by NASA)
- Performance analysis
  - **Insights** into performance bottlenecks

Execution time assessment

Interesting use cases

- Predicting program run time in **remote** environments or time-sensitive applications
  - Examples: firmware in cars or satellite (e.g. Kepler space telescope by NASA)
- Performance analysis
  - **Insights** into performance bottlenecks
    - Examples: Potential CPU pipeline stalling, GPU memory bank conflicts

Execution time assessment

Previous efforts

Execution time assessment

Previous efforts

- **Static** approaches

Execution time assessment

Previous efforts

- **Static** approaches
  - Throughput analysis: predicting the cycle counts for linear code (e.g. basic block, loop) statically

Execution time assessment

Previous efforts

- **Static** approaches
  - Throughput analysis: predicting the cycle counts for linear code (e.g. basic block, loop) statically
    - Examples: IACA, OSACA, uiCA, LLVM MCA, Ithemal

Execution time assessment

Previous efforts

- **Static** approaches
  - Throughput analysis: predicting the cycle counts for linear code (e.g. basic block, loop) statically
    - Examples: IACA, OSACA, uiCA, LLVM MCA, Ithemal
- **Dynamic** approaches
  - Cycle-accurate simulators / emulators

Execution time assessment

Previous efforts

- **Static** approaches
  - Throughput analysis: predicting the cycle counts for linear code (e.g. basic block, loop) statically
    - Examples: IACA, OSACA, uiCA, LLVM MCA, Ithemal
- **Dynamic** approaches
  - Cycle-accurate simulators / emulators
    - Examples: gem5, gpgpu-sim

Execution time assessment

Challenges

Static

Dynamic

Execution time assessment

Challenges

Static **Dynamic**

Execution time assessment

Challenges

Static

Dynamic

- Complete execution traces
- Higher fidelity on hardware details

Execution time assessment

Challenges

Static

- Poor handling on branches & function calls
  - Small scope (only few blocks)
- Lack of run-time information

Dynamic

- Complete execution traces
- Higher fidelity on hardware details

Execution time assessment

Challenges

Static

- Poor handling on branches & function calls
  - Small scope (only few blocks)
- Lack of run-time information

Dynamic

- Complete execution traces
- Higher fidelity on hardware details

Execution time assessment

Challenges

Precision

Low → High

Static

- Poor handling on branches & function calls
  - Small scope (only few blocks)
- Lack of run-time information
- Faster analysis speed (due to coarser granularity)
- Easier integration with other tools

Dynamic

- Complete execution traces
- Higher fidelity on hardware details

Fast ← Slow

Turnaround

Execution time assessment

Challenges

Precision

Low → High

Static

- Poor handling on branches & function calls
  - Small scope (only few blocks)
- Lack of run-time information
- Faster analysis speed (due to coarser granularity)
- Easier integration with other tools

Dynamic

- Complete execution traces
- Higher fidelity on hardware details
- Usually require non-trivial setup
- Slow simulation speed

Fast ← Slow

Turnaround

Execution time assessment

Challenges

Precision

Low → High

Static

- Poor handling on branches & function calls
  - Small scope (only few blocks)
- Lack of run-time information
- Faster analysis speed (due to coarser granularity)
- Easier integration with other tools

Dynamic

- Complete execution traces
- Higher fidelity on hardware details
- Usually require non-trivial setup
- Slow simulation speed

Fast ← Slow

Turnaround

Outline

Motivation

MCA Daemon (MCAD)

Future Plans

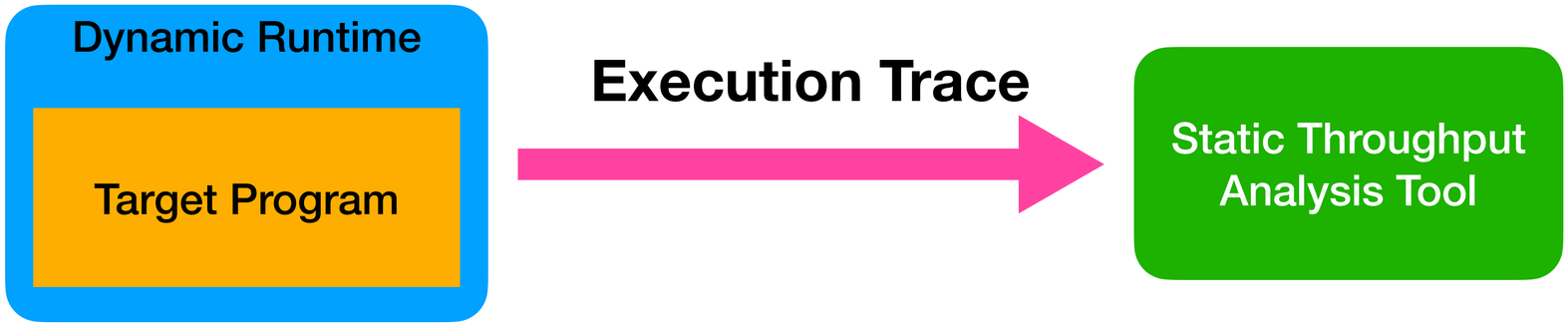
Epilogue

MCA Daemon (MCAD)

High-level concept

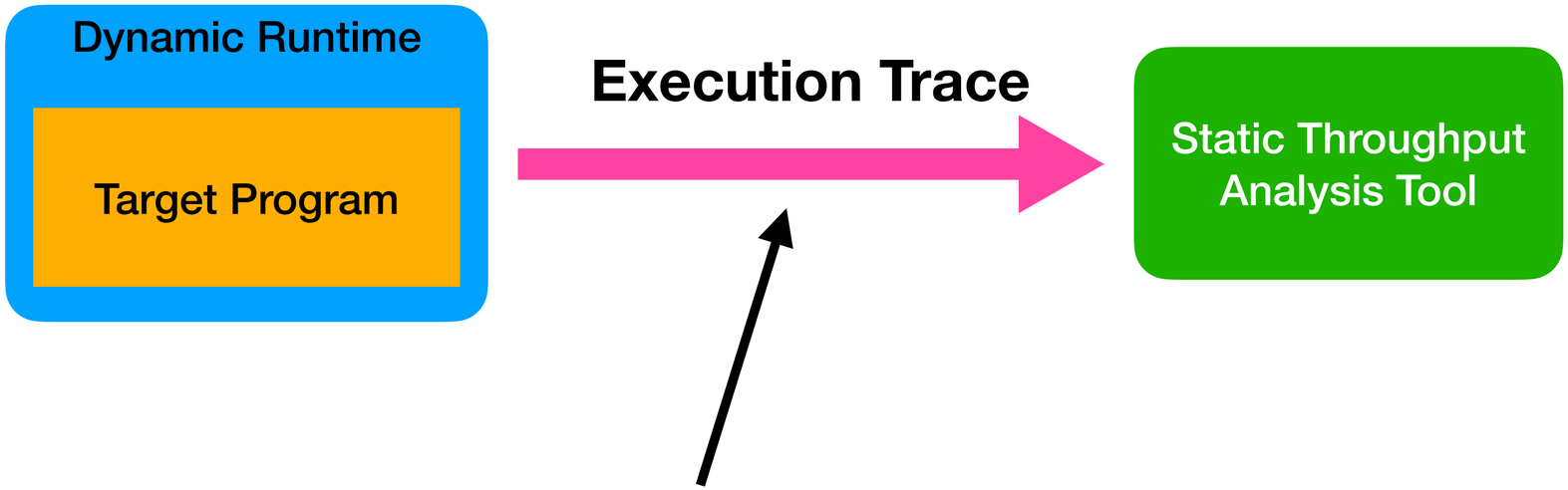
MCA Daemon (MCAD)

High-level concept



MCA Daemon (MCAD)

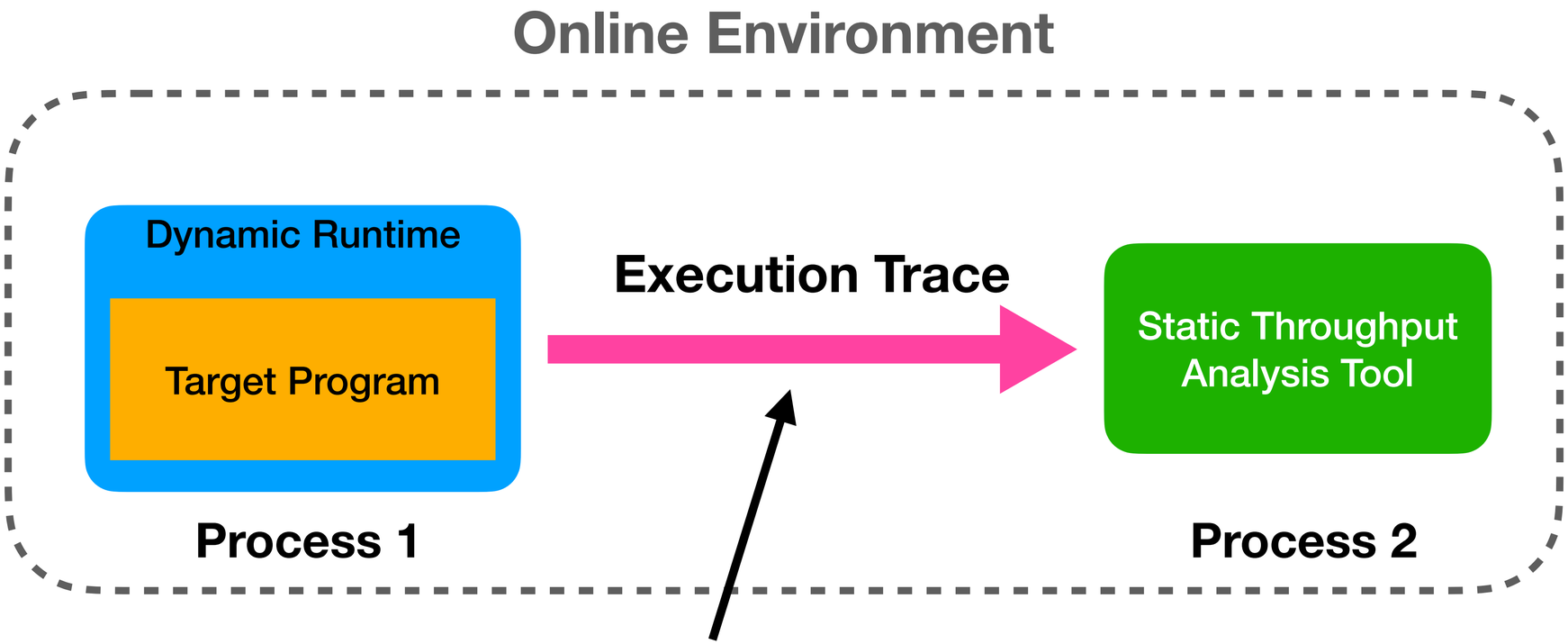
High-level concept



- The instructions that just got executed
- Run-time values (e.g. register values)

MCA Daemon (MCAD)

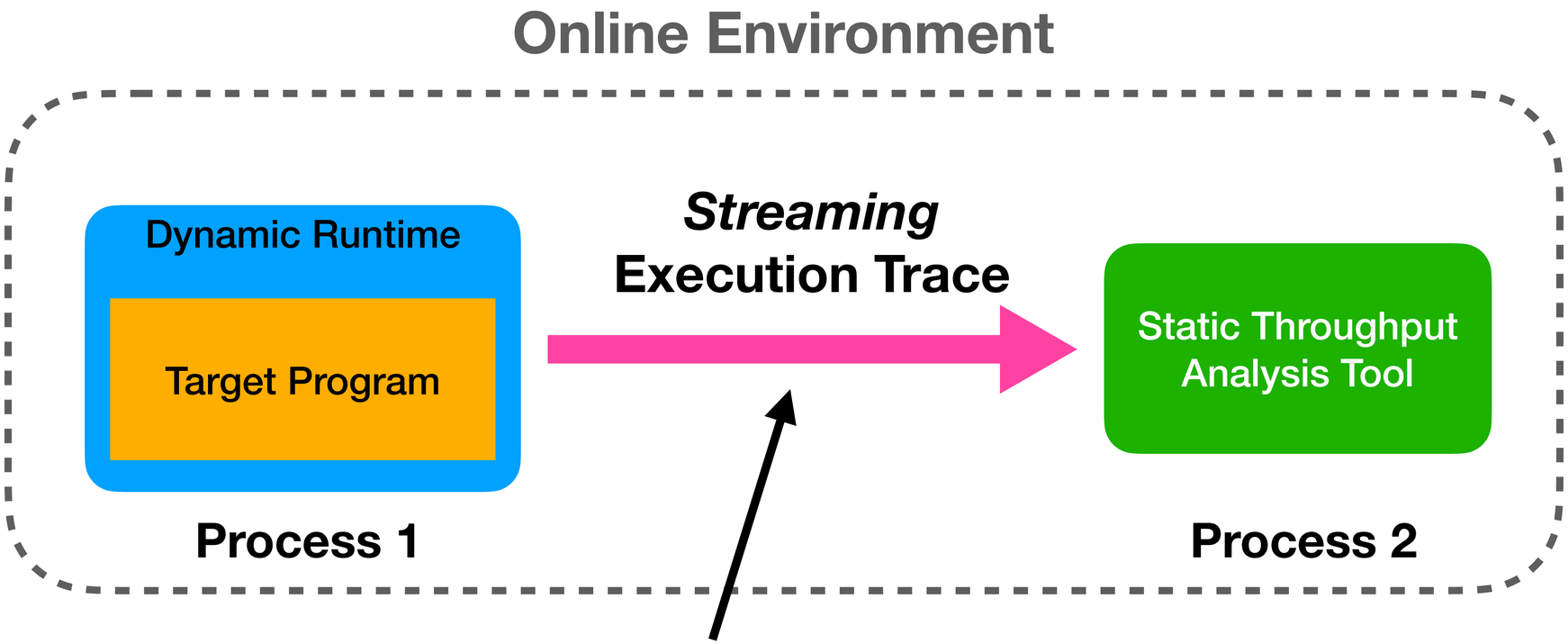
High-level concept



- The instructions that just got executed
- Run-time values (e.g. register values)

MCA Daemon (MCAD)

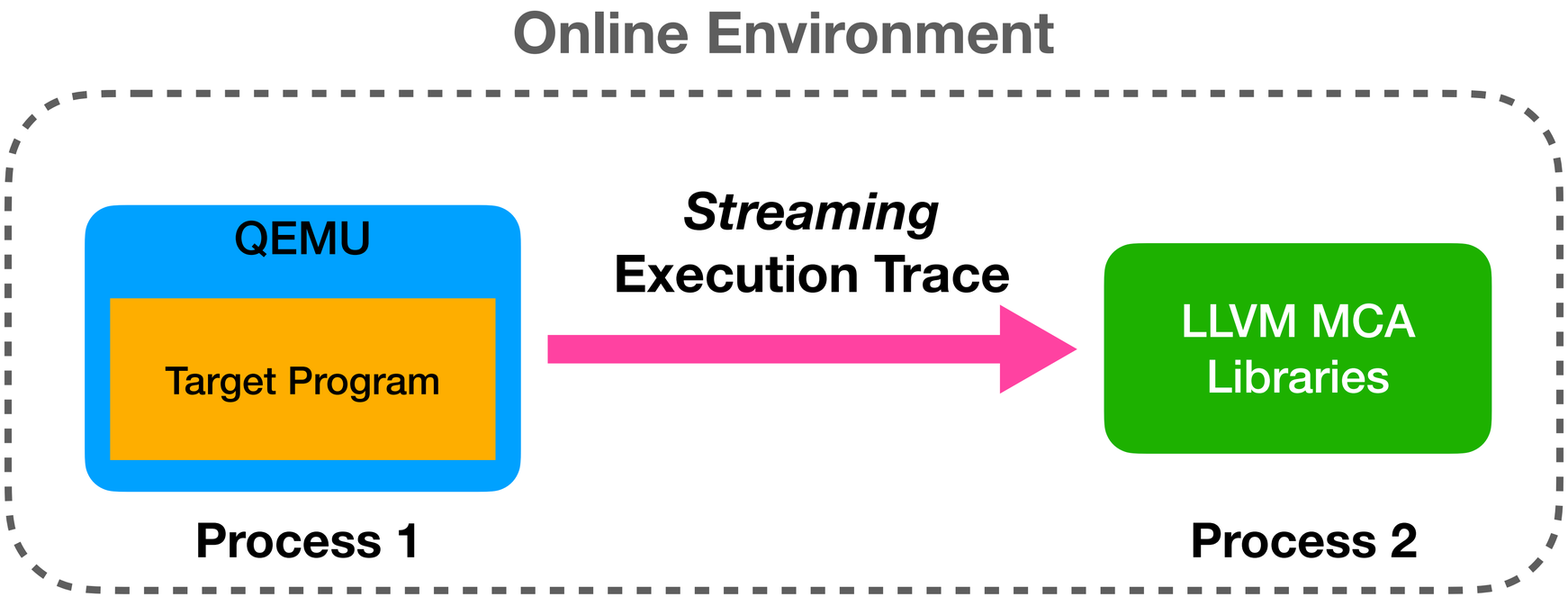
High-level concept



- The instructions that just got executed
- Run-time values (e.g. register values)

MCA Daemon (MCAD)

High-level concept



Introduction to LLVM MCA

Introduction to LLVM MCA

- A tool (llvm-mca) and **library** (libLLVMMCA) for predicting cycle counts and potential performance hazards in a sequence of assembly code

Introduction to LLVM MCA

- A tool (llvm-mca) and **library** (libLLVMMCA) for predicting cycle counts and potential performance hazards in a sequence of assembly code
- Using instruction scheduling data (e.g. instruction latency) provided by each LLVM target
  - New ISA (with proper scheduling info) can be supported *out of the box*

Introduction to LLVM MCA

- A tool (llvm-mca) and **library** (libLLVMMCA) for predicting cycle counts and potential performance hazards in a sequence of assembly code
- Using instruction scheduling data (e.g. instruction latency) provided by each LLVM target
  - New ISA (with proper scheduling info) can be supported *out of the box*
- Accounting for modern processor features: super scalar, out-of-order etc.

Introduction to LLVM MCA

- A tool (llvm-mca) and **library** (libLLVMMCA) for predicting cycle counts and potential performance hazards in a sequence of assembly code
- Using instruction scheduling data (e.g. instruction latency) provided by each LLVM target
  - New ISA (with proper scheduling info) can be supported *out of the box*
- Accounting for modern processor features: super scalar, out-of-order etc.
- Implemented via lightweight simulation
  - Abstract real CPU pipeline stages into a small handful of stages

Introduction to MCA

An example

test/tools/llvm-mca/X86/BtVer2/dot-product.s

```
vmulps    %xmm0, %xmm1, %xmm2
vhaddps   %xmm2, %xmm2, %xmm3
vhaddps   %xmm3, %xmm3, %xmm4
```

Introduction to MCA

An example

test/tools/llvm-mca/X86/BtVer2/dot-product.s

```
vmulps    %xmm0, %xmm1, %xmm2
vhaddps   %xmm2, %xmm2, %xmm3
vhaddps   %xmm3, %xmm3, %xmm4
```

```
llvm-mca -mtriple=x86_64 -mcpu=btver2 \
         -iterations=300 dot-products.s
```

Introduction to MCA

An example

test/tools/llvm-mca/X86/BtVer2/dot-product.s

```
vmulps    %xmm0, %xmm1, %xmm2
vhaddps   %xmm2, %xmm2, %xmm3
vhaddps   %xmm3, %xmm3, %xmm4
```

```
llvm-mca -mtriple=x86_64 -mcpu=btver2 \
         -iterations=300 dot-products.s
```

Summary

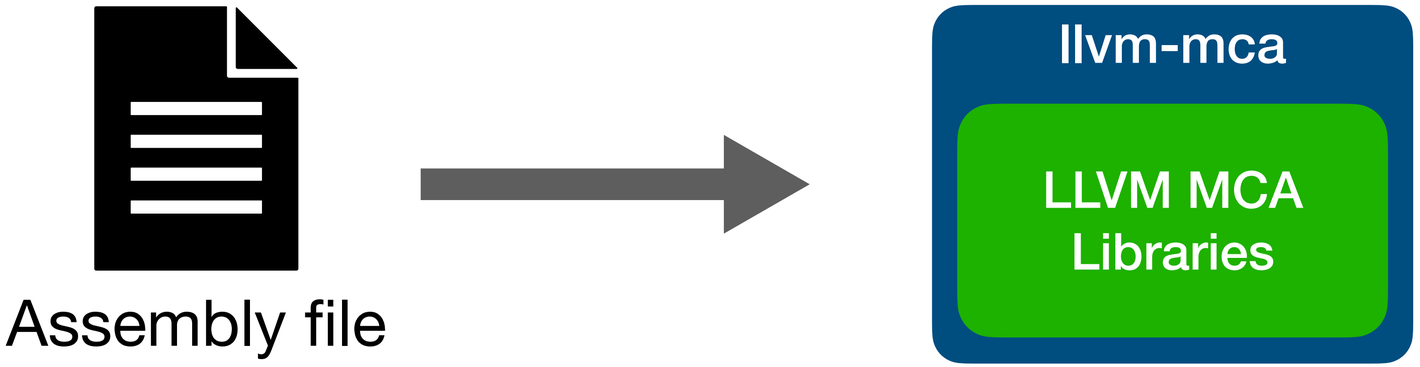
```
Iterations:          300
Instructions:        900
Total Cycles:        610
Total uOps:          900

Dispatch Width:      2
uOps Per Cycle:      1.48
IPC:                 1.48
Block RThroughput:   2.0
```

llvm-mca

MCAD

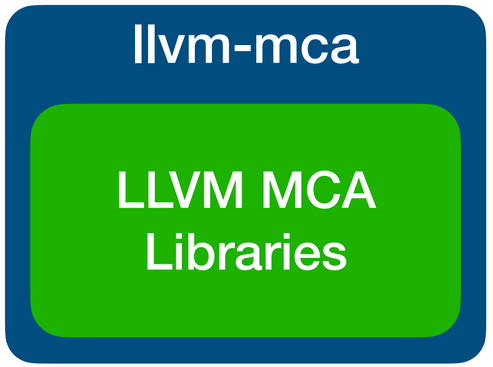
llvm-mca



MCAD

llvm-mca

Assembly file



MCAD

Process 1

Streaming
Execution Trace

Process 2

MCA Daemon (MCAD)

Highlights

MCA Daemon (MCAD)

Highlights

- Combine the advantages of dynamic & static throughput analysis

MCA Daemon (MCAD)

Highlights

- Combine the advantages of dynamic & static throughput analysis
- Augment the analysis region *beyond* basic blocks
  - MCAD is able to analyze the **entire** program execution trace

MCA Daemon (MCAD)

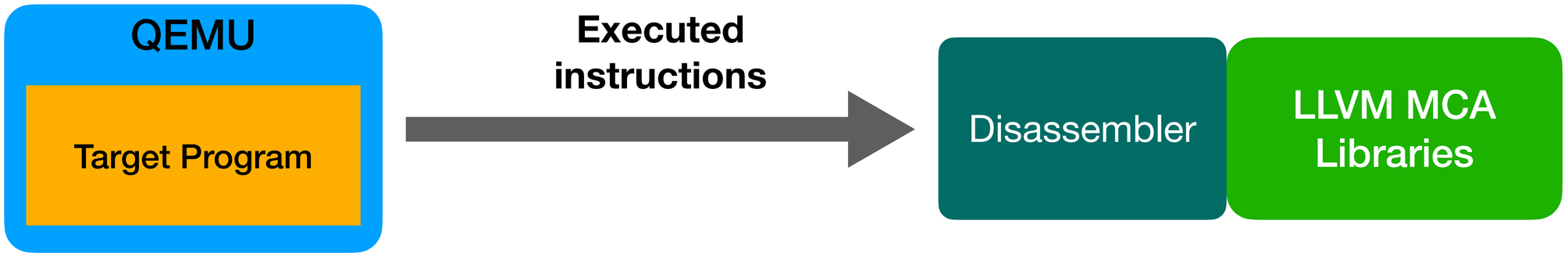
Highlights

- Combine the advantages of dynamic & static throughput analysis
- Augment the analysis region *beyond* basic blocks
  - MCAD is able to analyze the **entire** program execution trace
- Throughput analysis is happening **in parallel** / **on-the-fly** with the target program execution

Implementation

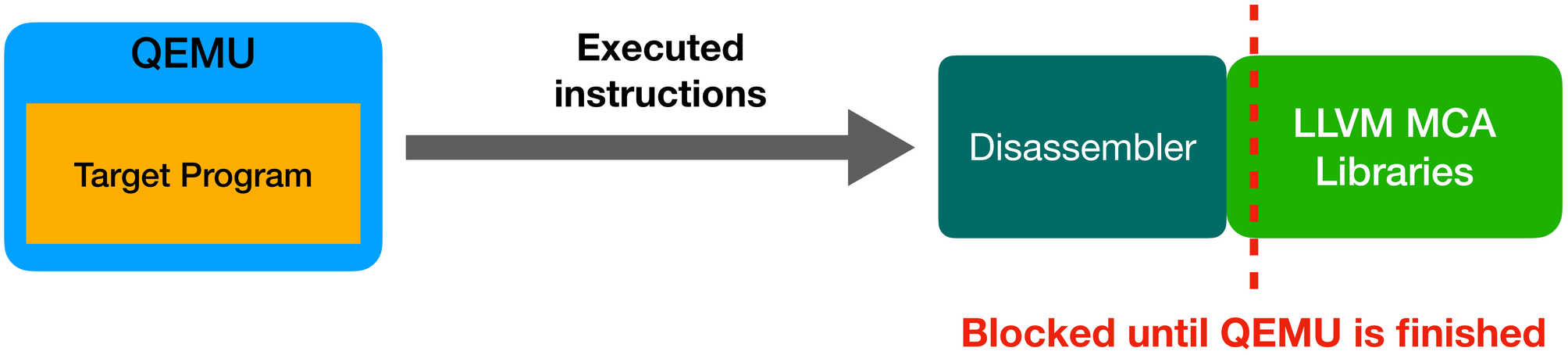
Analyze execution traces using MCA

Using unmodified MCA libraries



Analyze execution traces using MCA

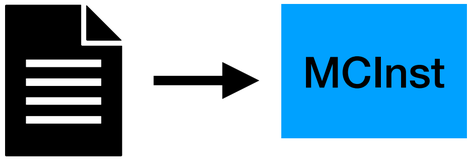
Challenge: Sequential workflow



MCA internal

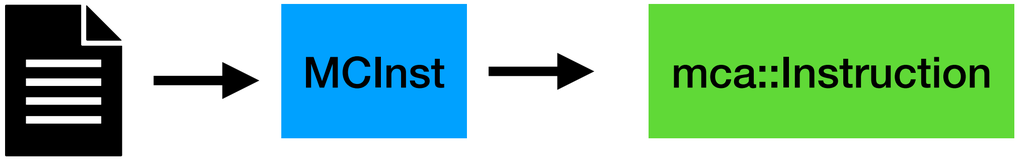
Assembly file

MCA internal



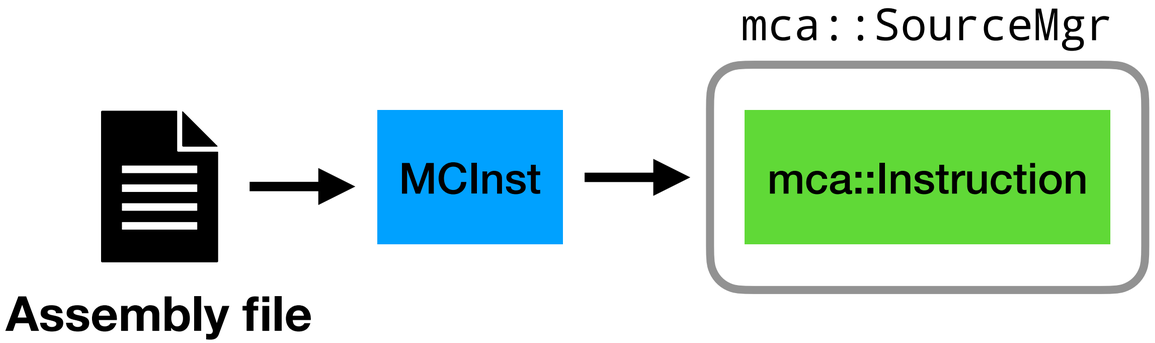
Assembly file

MCA internal

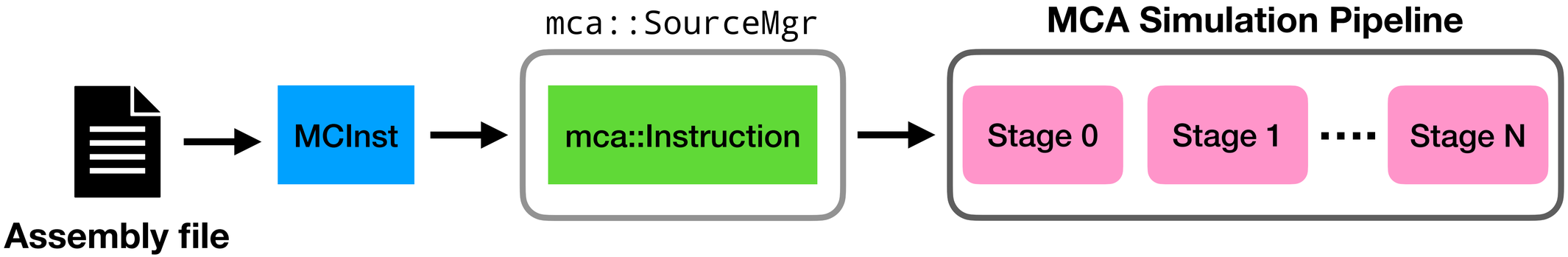


Assembly file

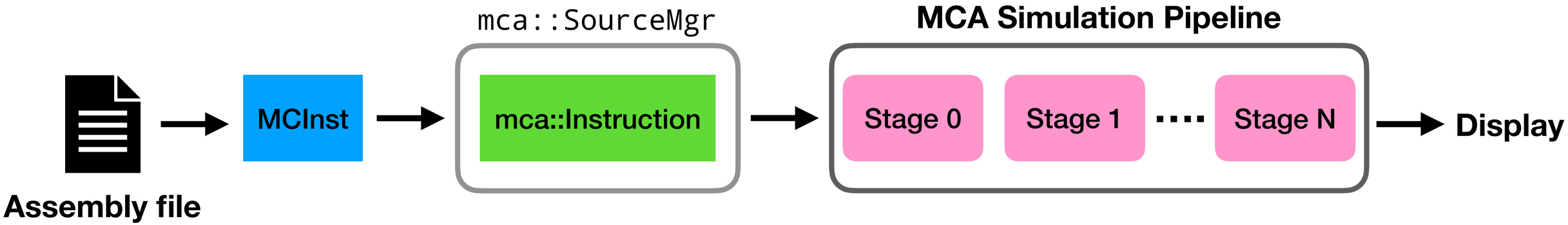
MCA internal



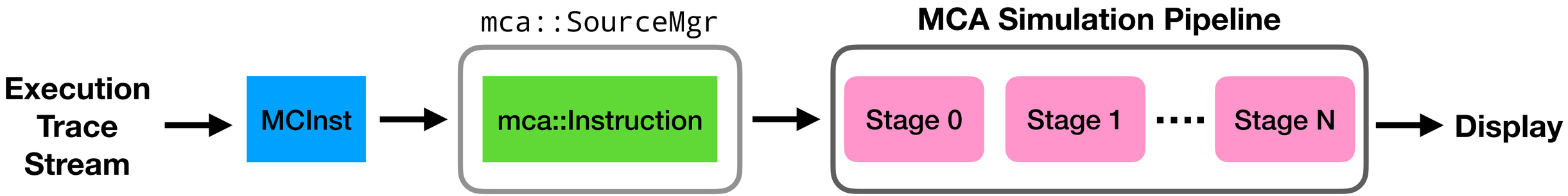
MCA internal



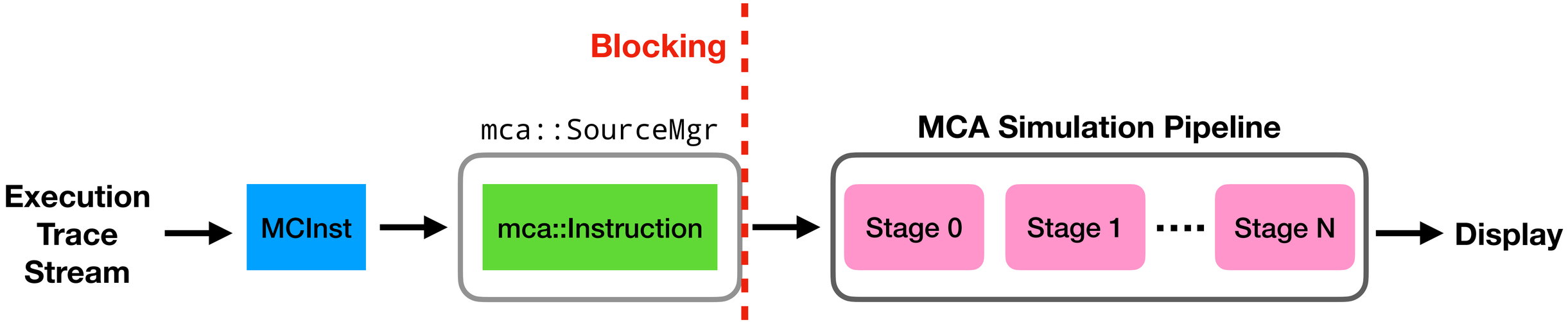
MCA internal



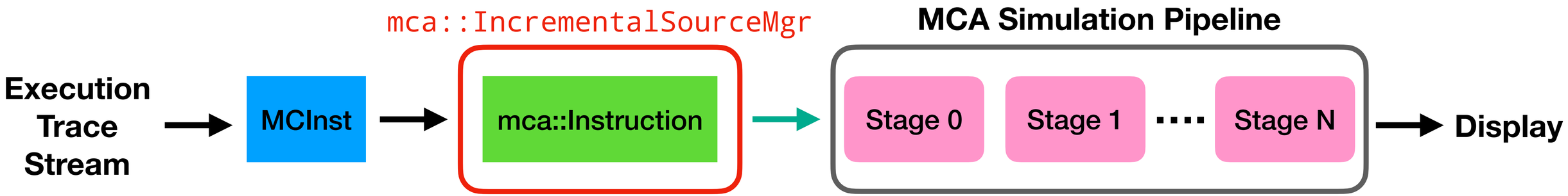
MCA with execution trace stream as input



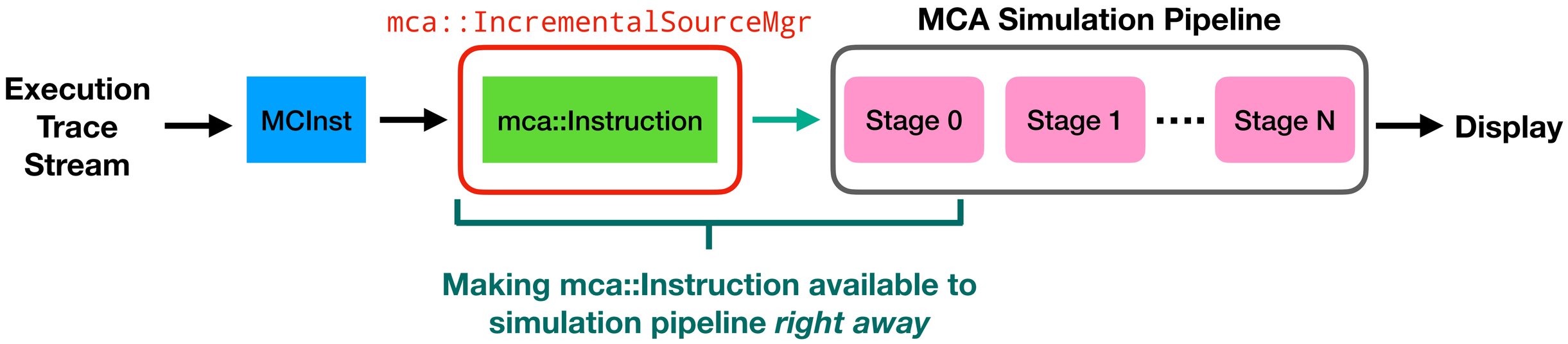
MCA with execution trace stream as input



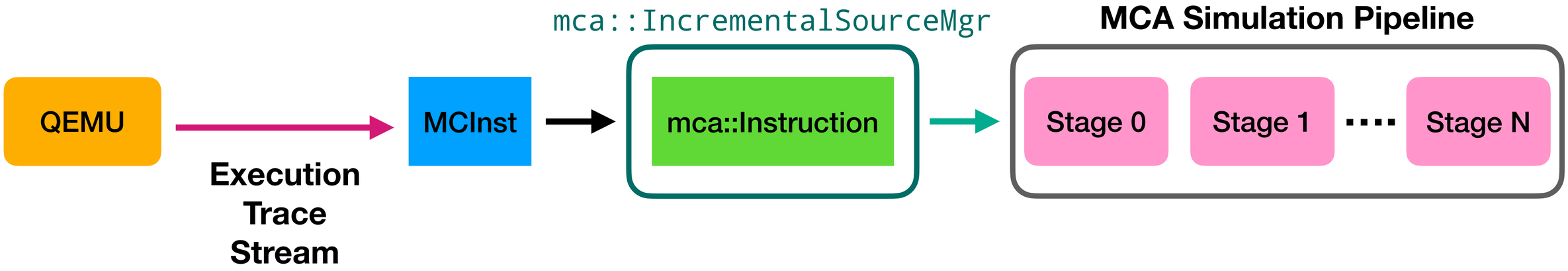
Incremental SourceMgr



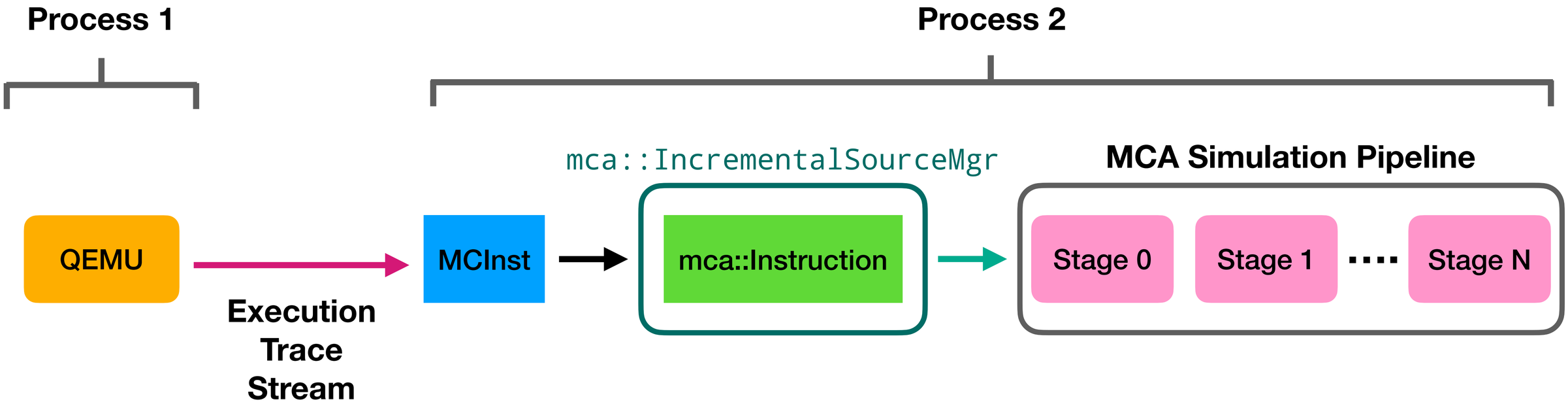
Incremental SourceMgr



Incremental SourceMgr

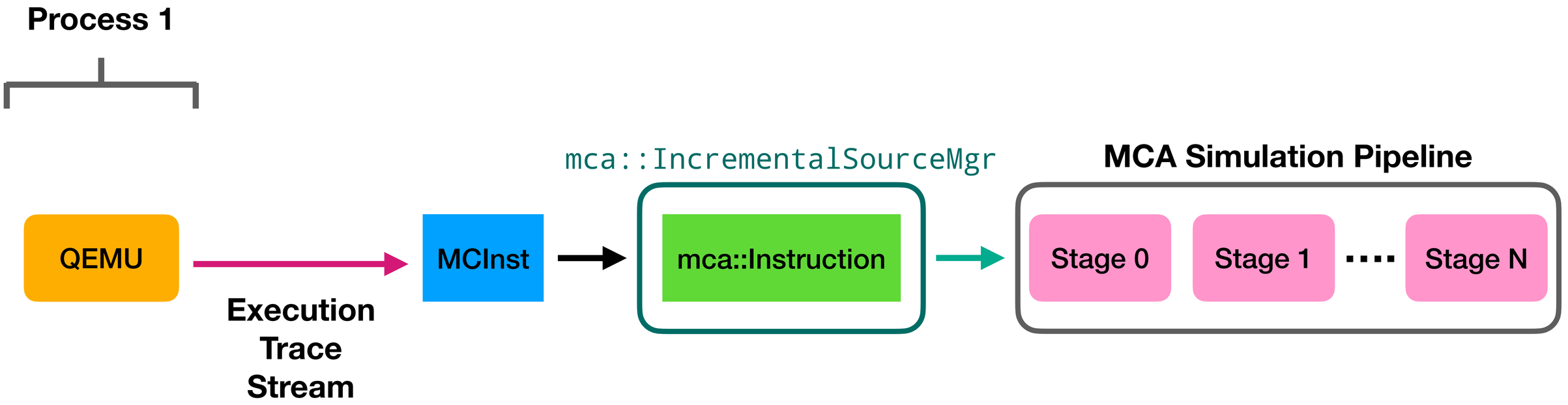


Incremental SourceMgr



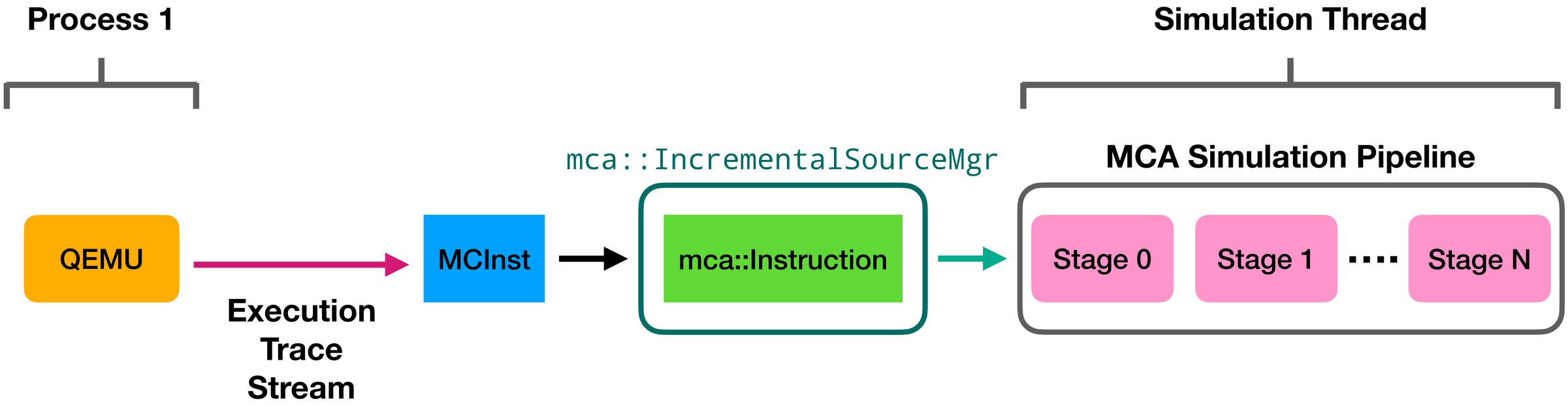
Incremental SourceMgr

Implement with threads



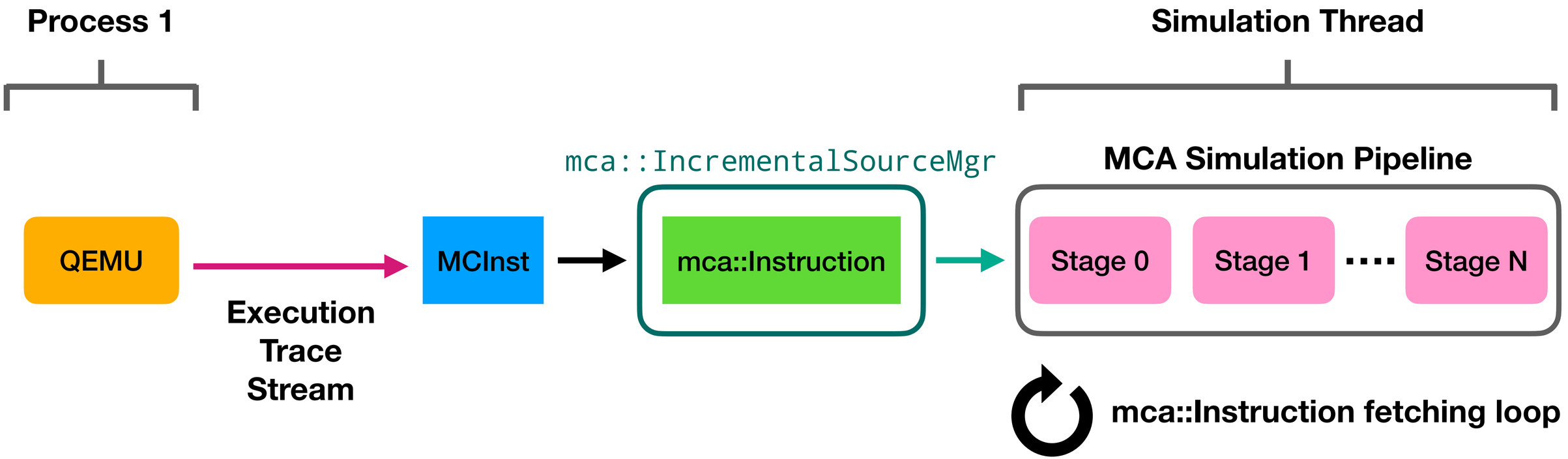
Incremental SourceMgr

Implement with threads



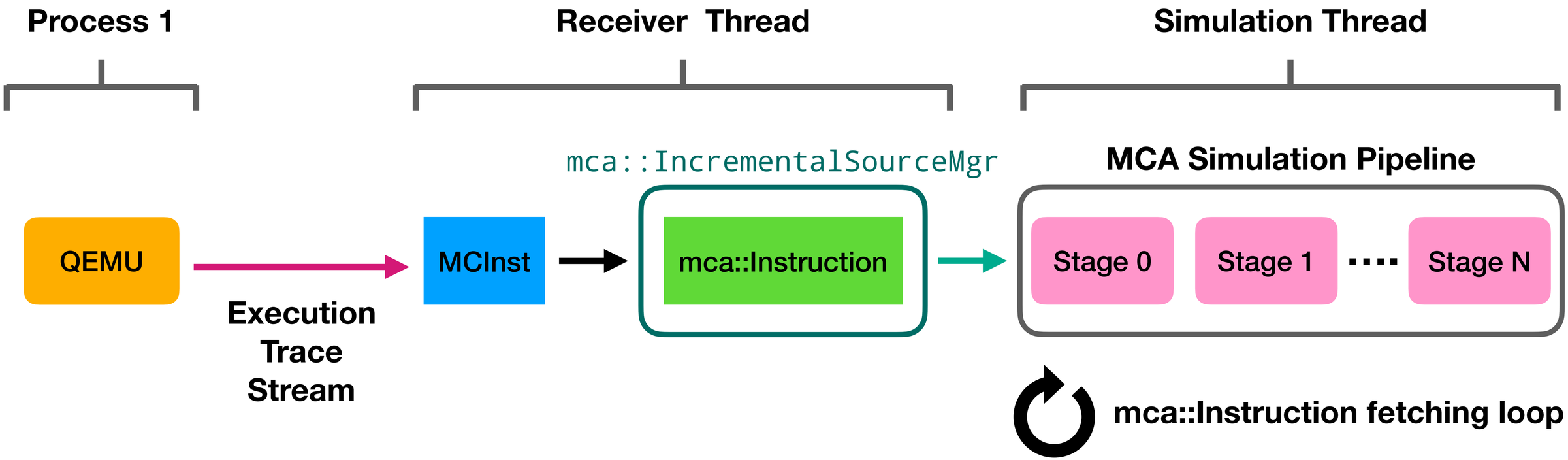
Incremental SourceMgr

Implement with threads



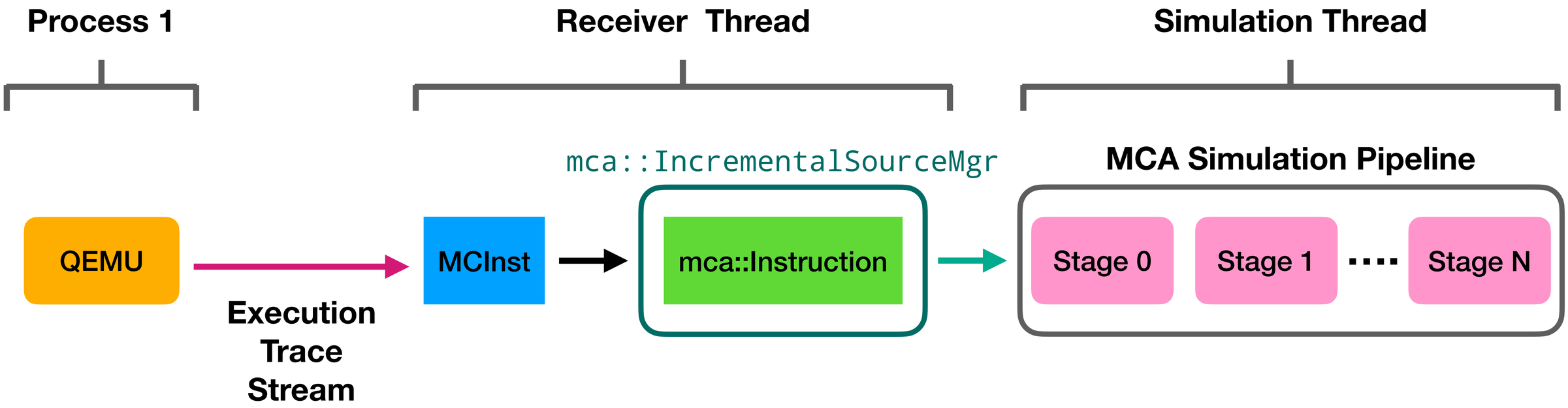
Incremental SourceMgr

Implement with threads



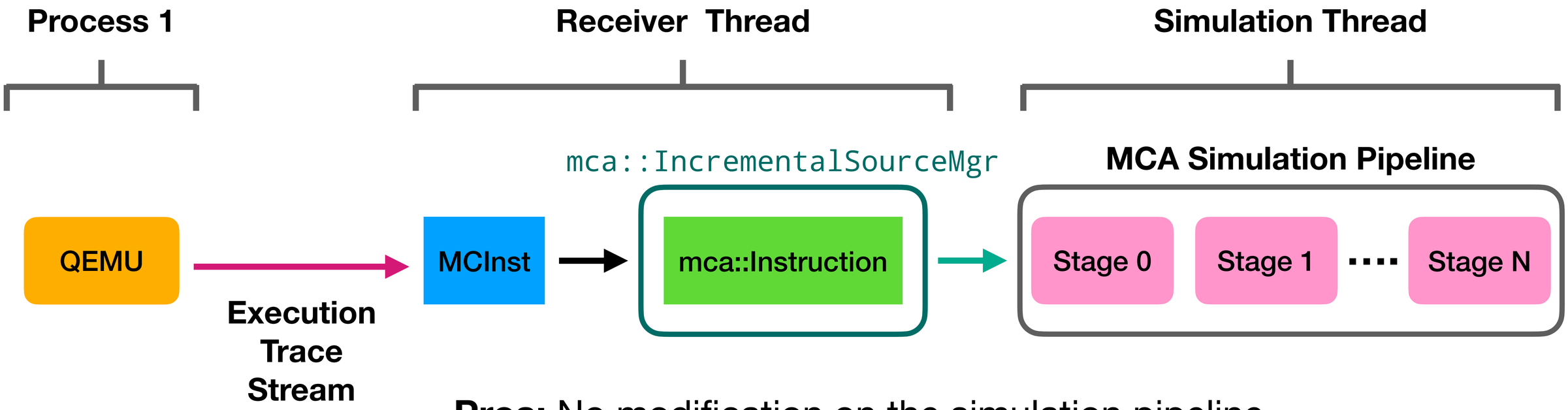
Incremental SourceMgr

Implement with threads: Pros & Cons



Incremental SourceMgr

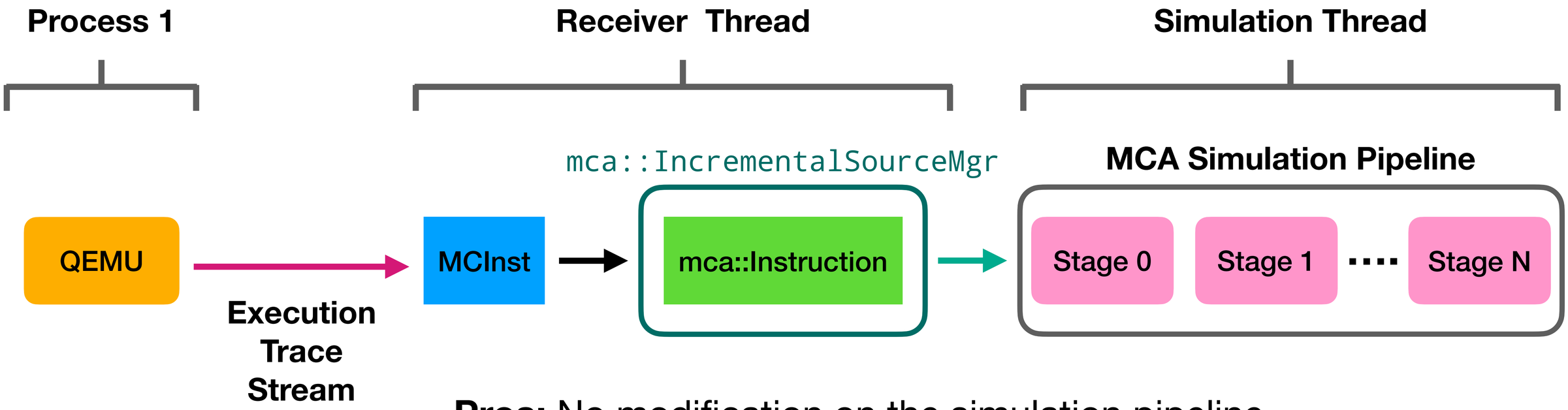
Implement with threads: Pros & Cons



Pros: No modification on the simulation pipeline

Incremental SourceMgr

Implement with threads: Pros & Cons

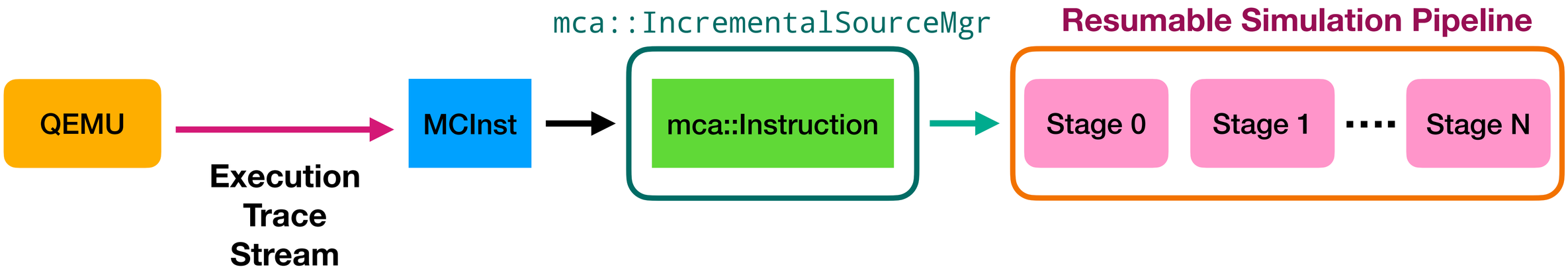


Pros: No modification on the simulation pipeline

Cons: To use IncrementalSourceMgr, you *have to* use threads

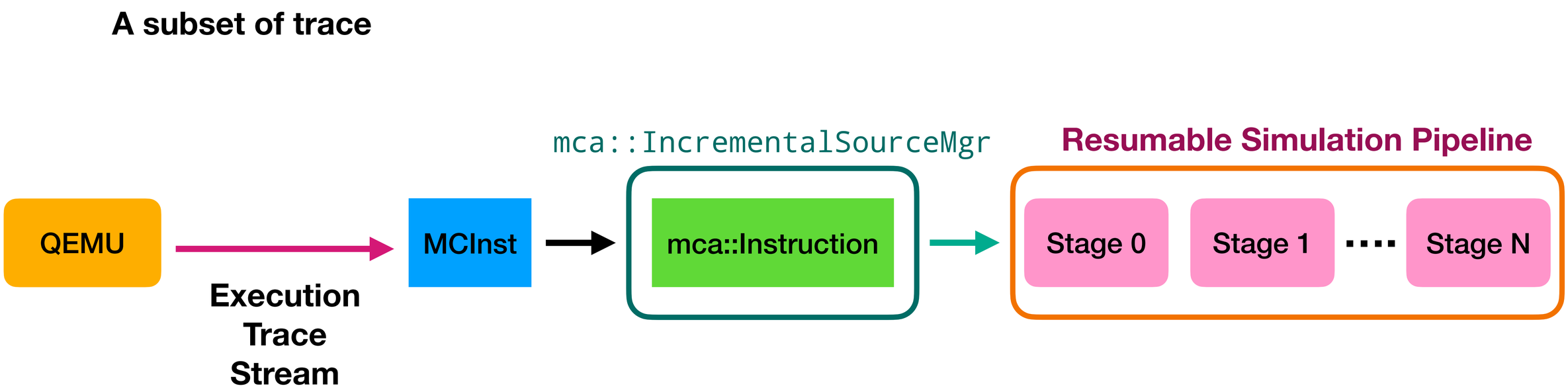
Incremental SourceMgr

Better solution: Resumable simulation pipeline



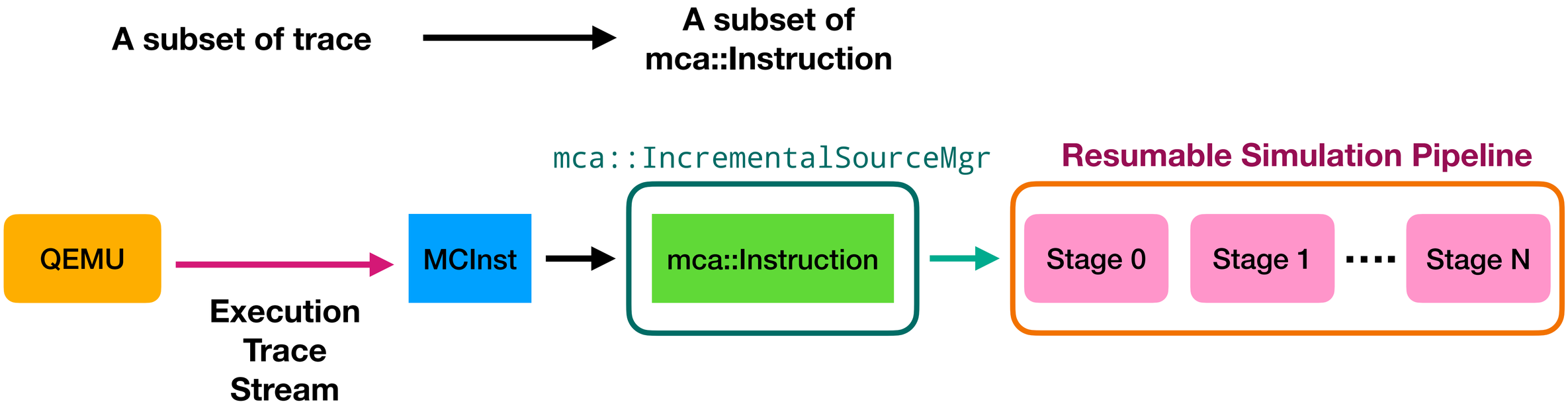
Incremental SourceMgr

Better solution: Resumable simulation pipeline



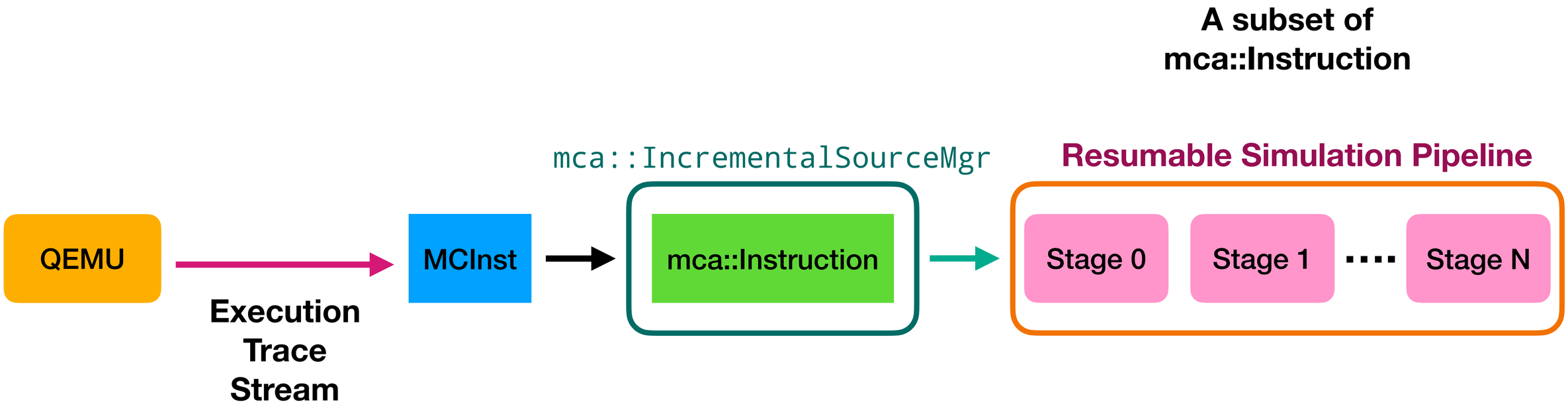
Incremental SourceMgr

Better solution: Resumable simulation pipeline



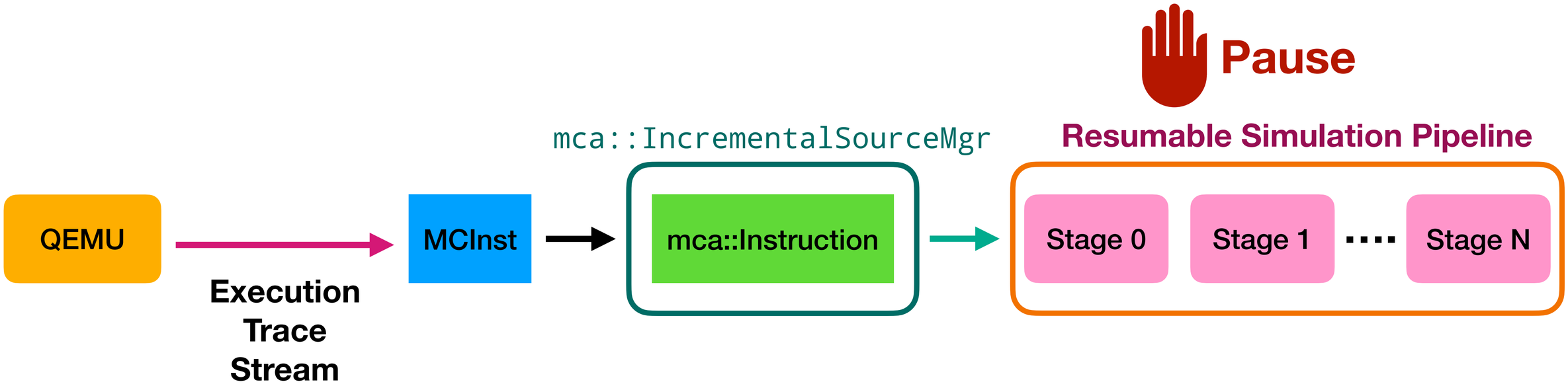
Incremental SourceMgr

Better solution: Resumable simulation pipeline



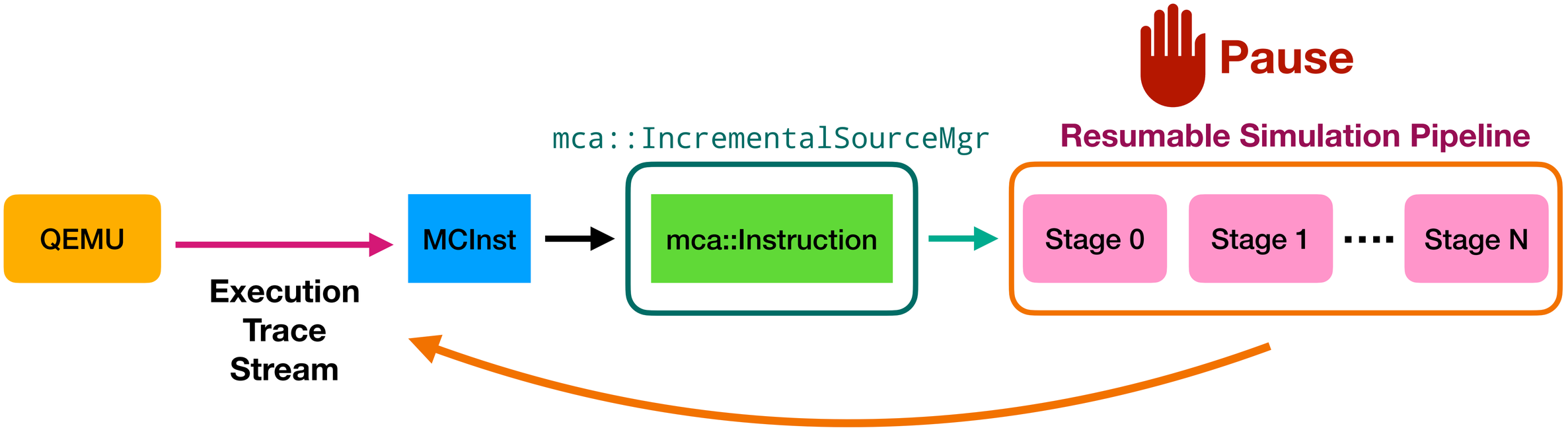
Incremental SourceMgr

Better solution: Resumable simulation pipeline



Incremental SourceMgr

Better solution: Resumable simulation pipeline



Resumable simulation pipeline

Resumable simulation pipeline

- **Save** (and restore) the analysis state from *previous* subset of instructions

Resumable simulation pipeline

- **Save** (and restore) the analysis state from *previous* subset of instructions
- Threads are **not required** when using IncrementalSourceMgr + resumable pipeline

Resumable simulation pipeline

- **Save** (and restore) the analysis state from *previous* subset of instructions
- Threads are **not required** when using IncrementalSourceMgr + resumable pipeline
  - Much easier to integrate into other uses

Resumable simulation pipeline

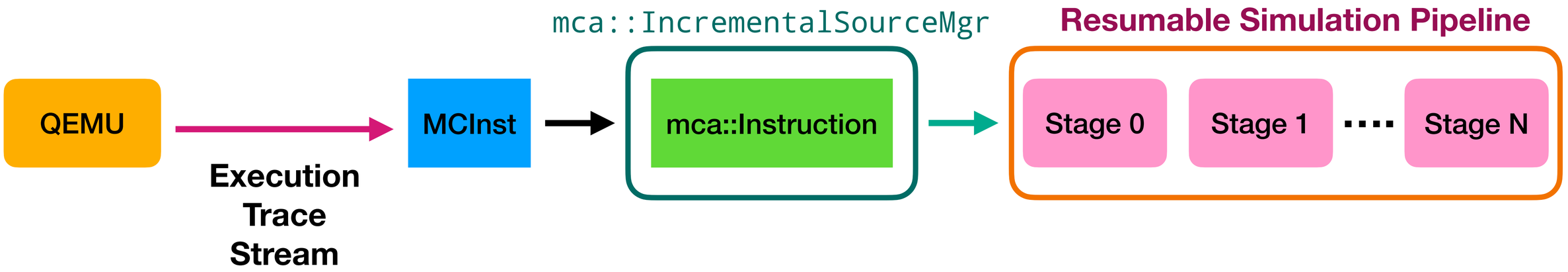
- **Save** (and restore) the analysis state from *previous* subset of instructions
- Threads are **not required** when using IncrementalSourceMgr + resumable pipeline
  - Much easier to integrate into other uses
  - You can still wrap resumable pipeline with a thread

Resumable simulation pipeline

- **Save** (and restore) the analysis state from *previous* subset of instructions
- Threads are **not required** when using IncrementalSourceMgr + resumable pipeline
  - Much easier to integrate into other uses
  - You can still wrap resumable pipeline with a thread
- Minor downside: Modifications on the simulation pipeline

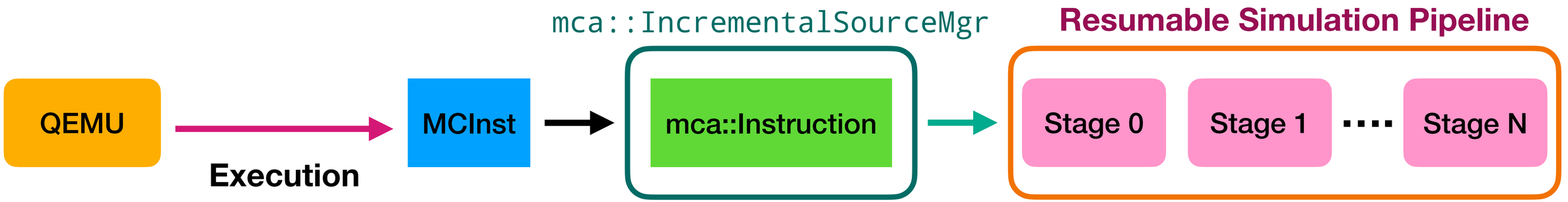
Incremental SourceMgr + Resumable pipeline

Put into real actions



Incremental SourceMgr + Resumable pipeline

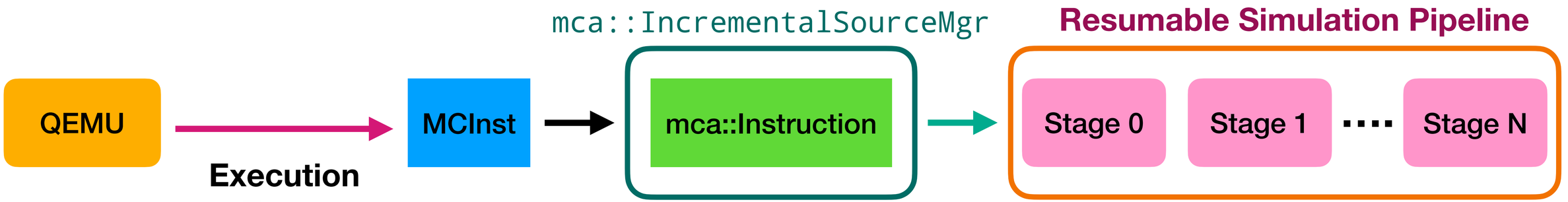
Put into real actions



- A program that continuously reads from an I/O device

Incremental SourceMgr + Resumable pipeline

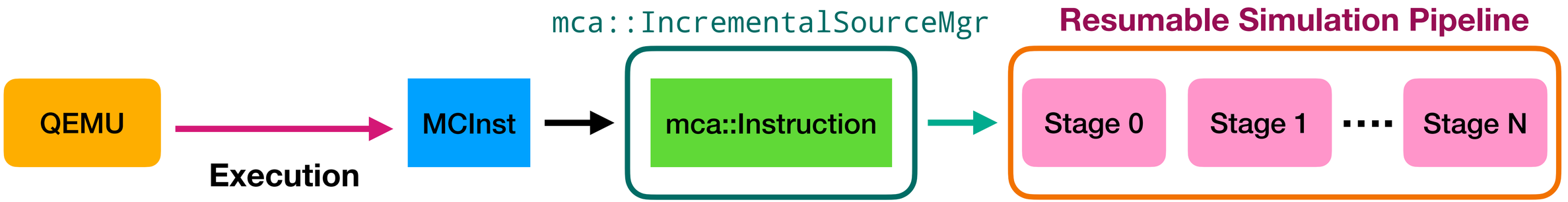
Put into real actions



- A program that continuously reads from an I/O device
- Only record the user space traces

Incremental SourceMgr + Resumable pipeline

Put into real actions



- A program that continuously reads from an I/O device
- Only record the user space traces
- Collected **~1 million** instructions

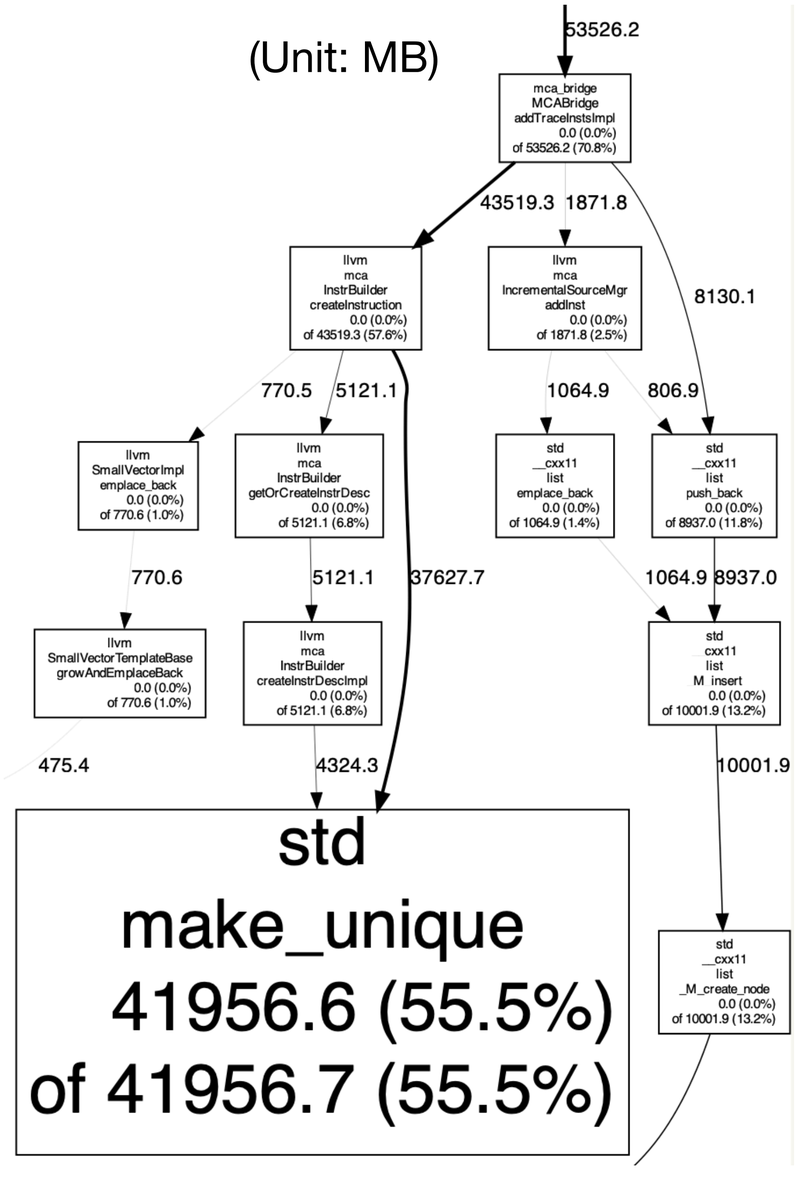
Challenge

Challenge

- Created a significant amount of **memory footprint**

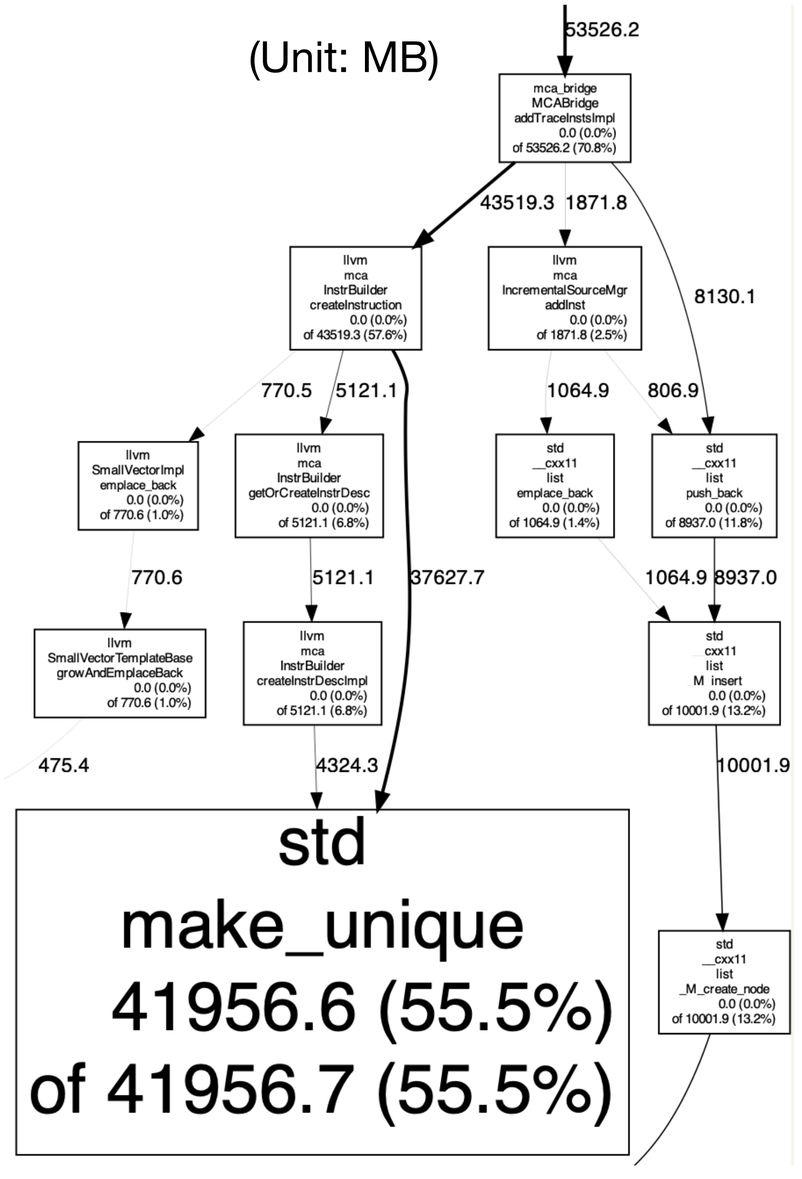
Challenge

- Created a significant amount of **memory footprint**



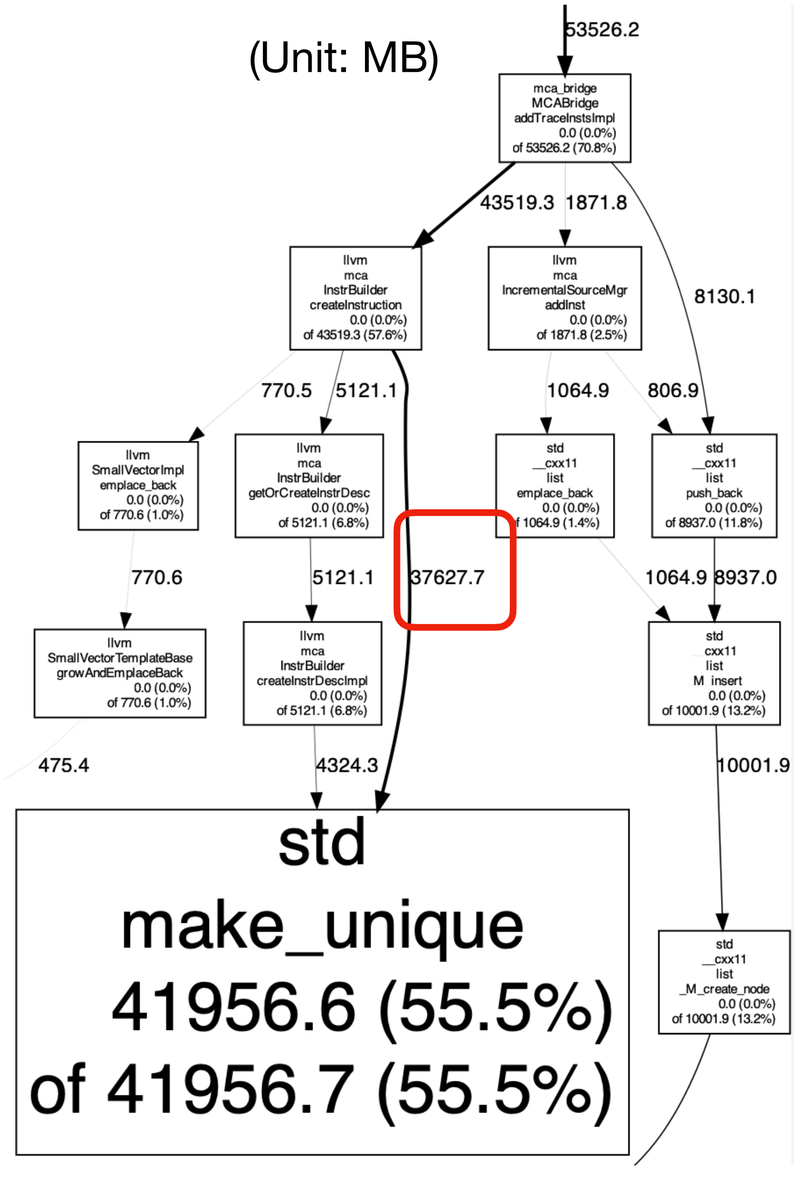
Challenge

- Created a significant amount of **memory footprint**
- Bottleneck: **~37GB** of accumulated (virtual) memory was allocated by `mca::InstrBuilder::createInstruction`



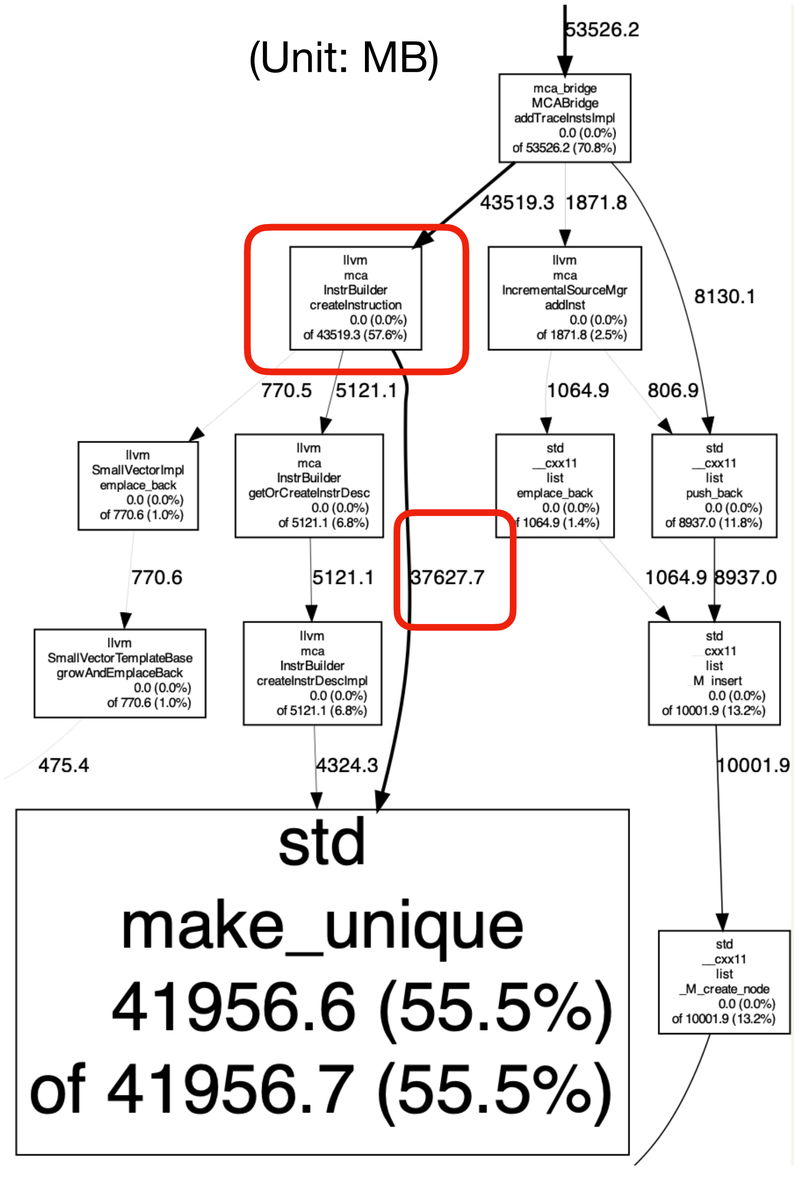
Challenge

- Created a significant amount of **memory footprint**
- Bottleneck: **~37GB** of accumulated (virtual) memory was allocated by `mca::InstrBuilder::createInstruction`



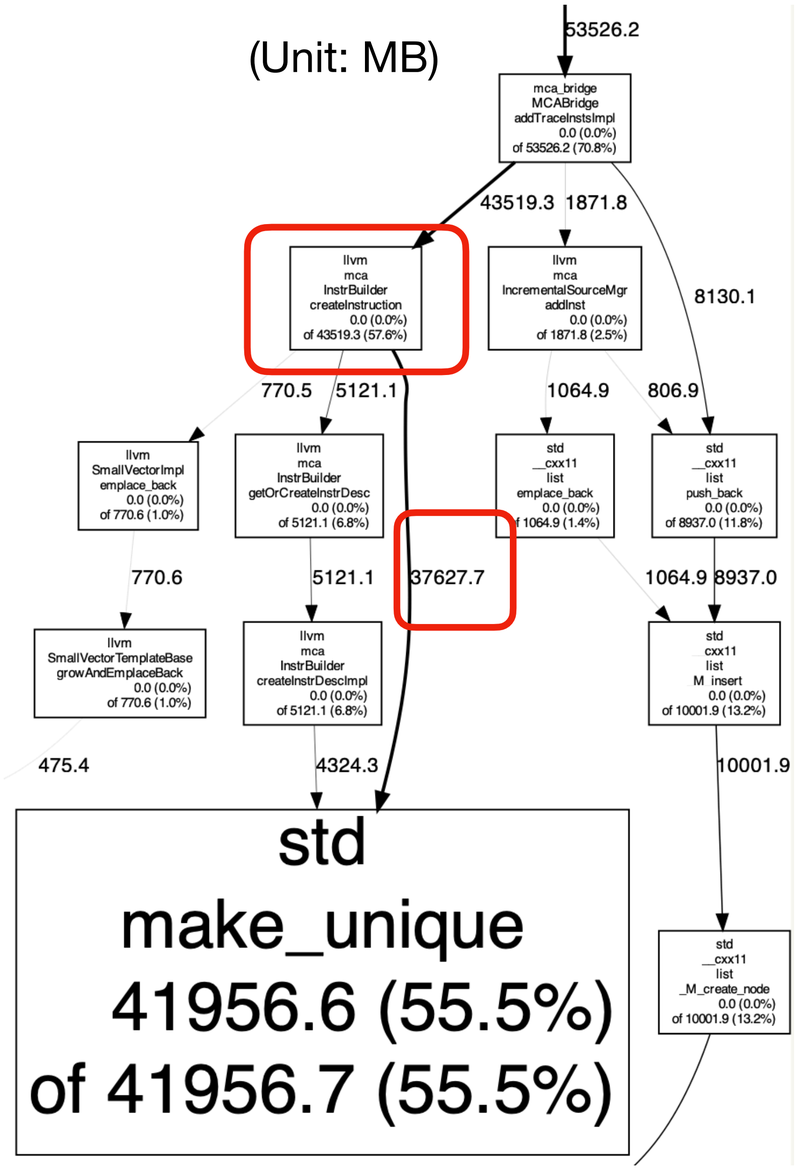
Challenge

- Created a significant amount of **memory footprint**
- Bottleneck: **~37GB** of accumulated (virtual) memory was allocated by `mca::InstrBuilder::createInstruction`



Challenge

- Created a significant amount of **memory footprint**
- Bottleneck: **~37GB** of accumulated (virtual) memory was allocated by `mca::InstrBuilder::createInstruction`



Large memory footprint

Root cause

Large memory footprint

Root cause

- Most of the translated `mca::Instruction` objects are **never** deallocated until the simulation is finished

Large memory footprint

Root cause

- Most of the translated `mca::Instruction` objects are **never** deallocated until the simulation is finished
- `mca::Instruction` is also used for tracking simulation state, so it's hard to make it immutable

Large memory footprint

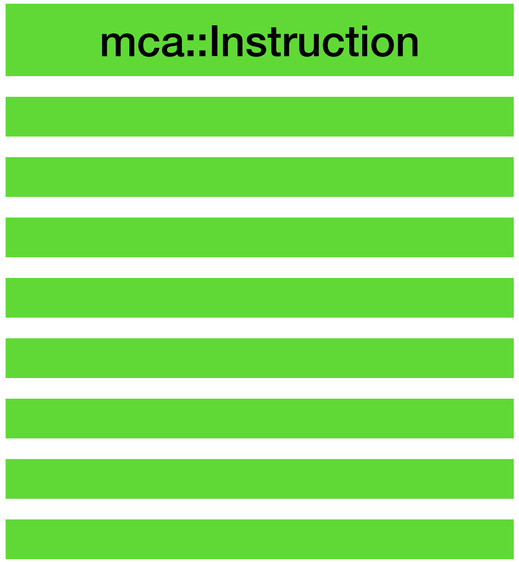
Root cause

- Most of the translated `mca::Instruction` objects are **never** deallocated until the simulation is finished
- `mca::Instruction` is also used for tracking simulation state, so it's hard to make it immutable
- Doesn't scale really well with large input (recall: ~1 million instructions)

Large memory footprint

Observation

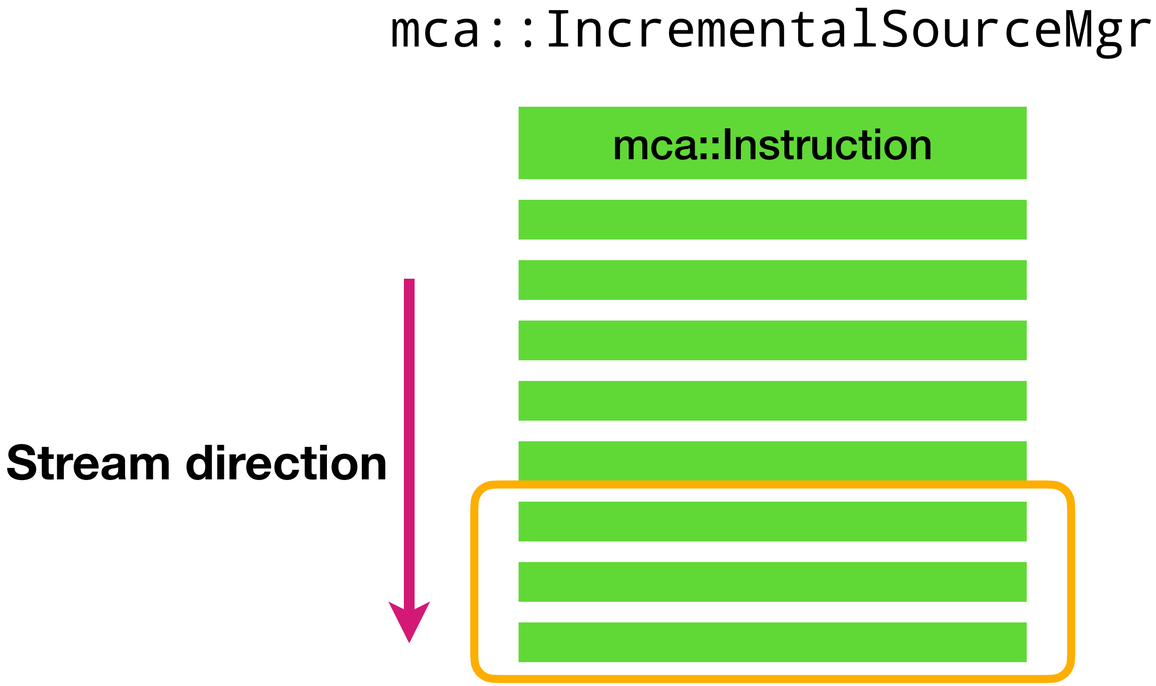
`mca::IncrementalSourceMgr`



Resumable Simulation Pipeline

Large memory footprint

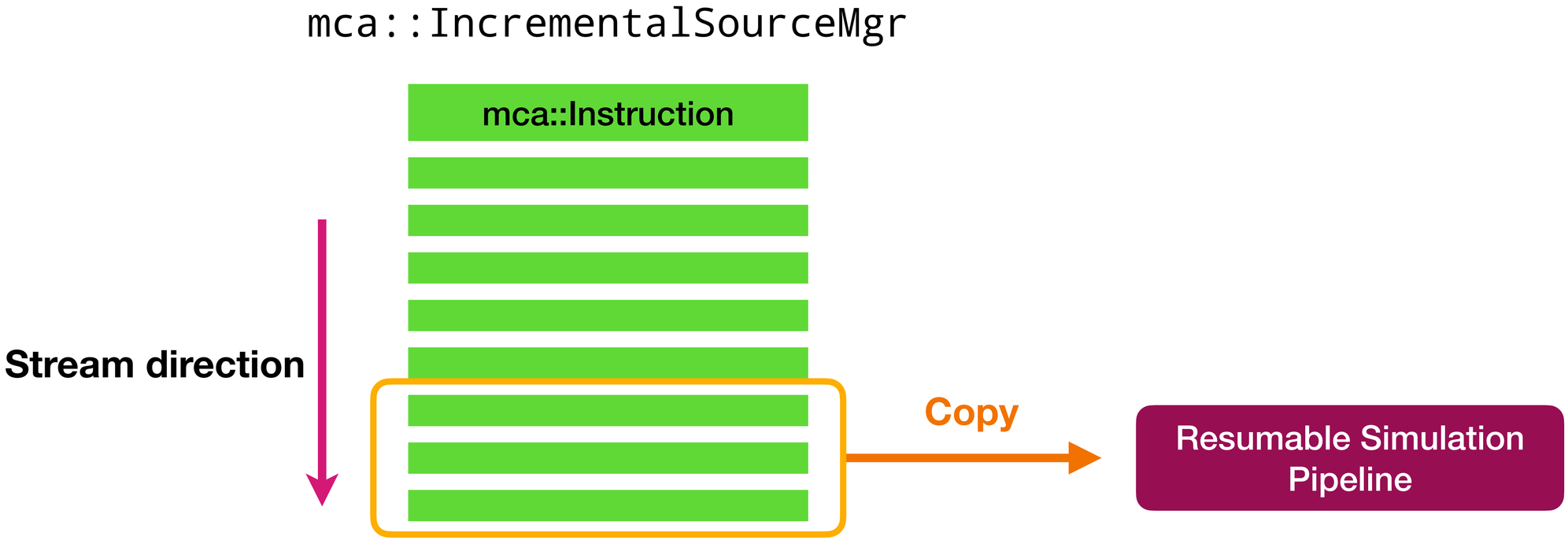
Observation



Resumable Simulation Pipeline

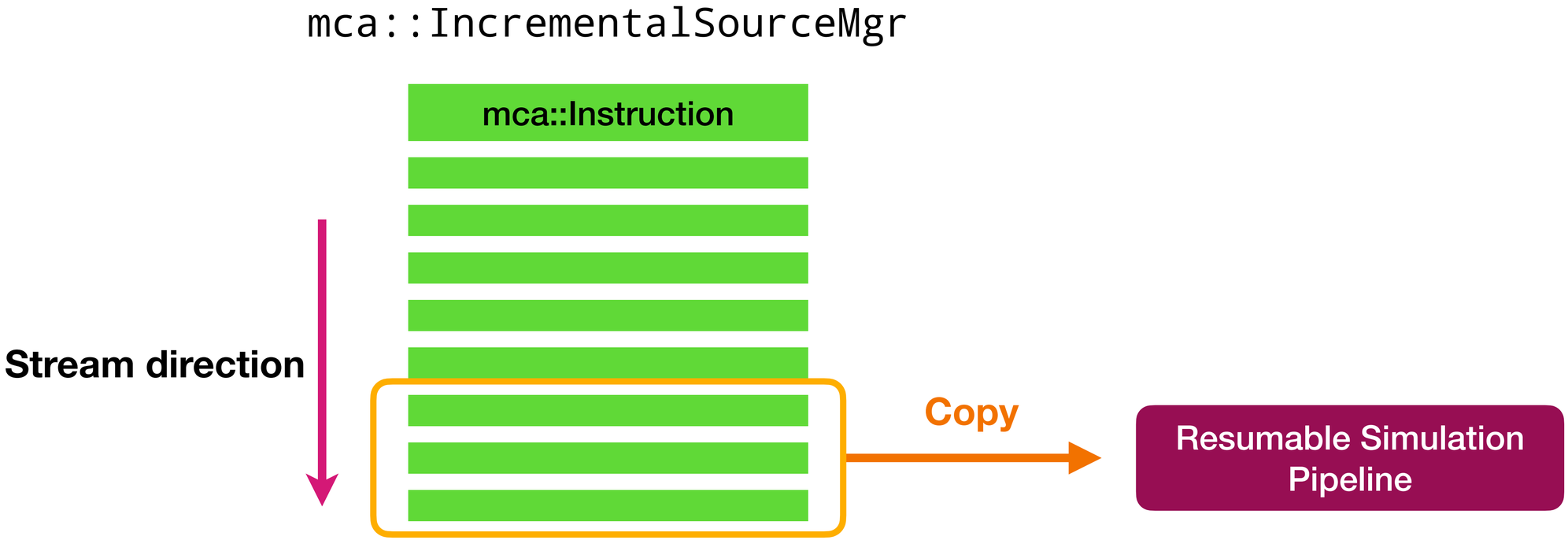
Large memory footprint

Observation



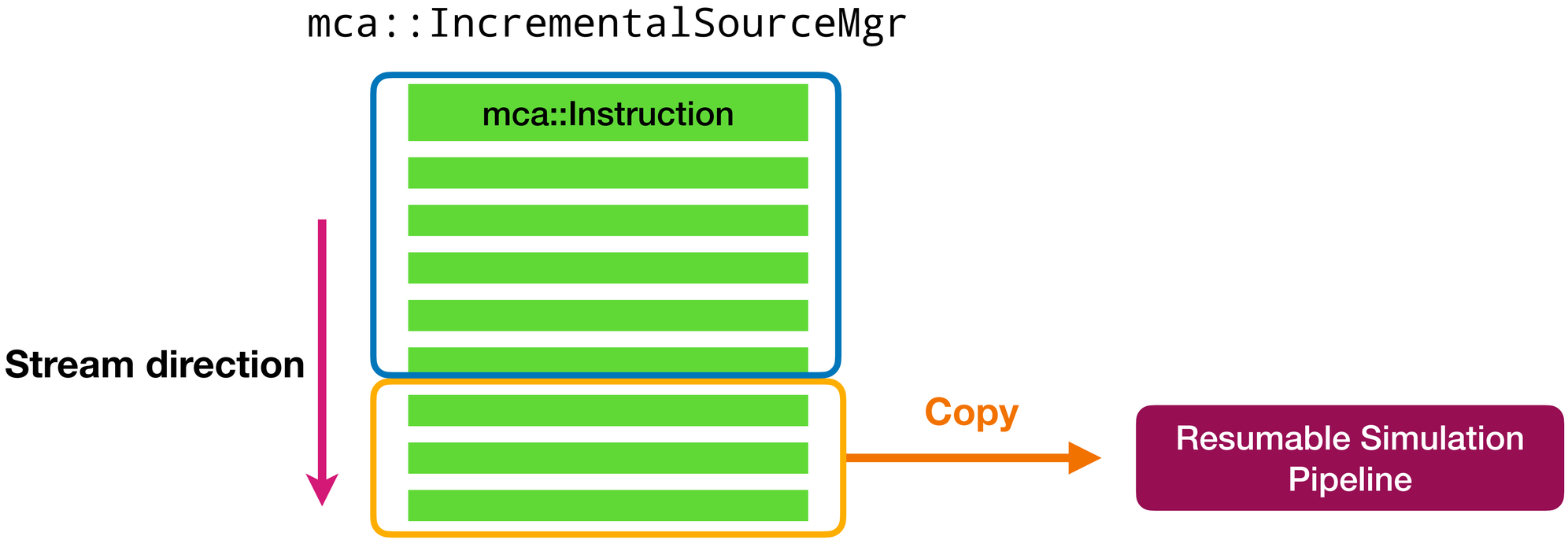
Large memory footprint

Solution: Recycling mca::Instruction



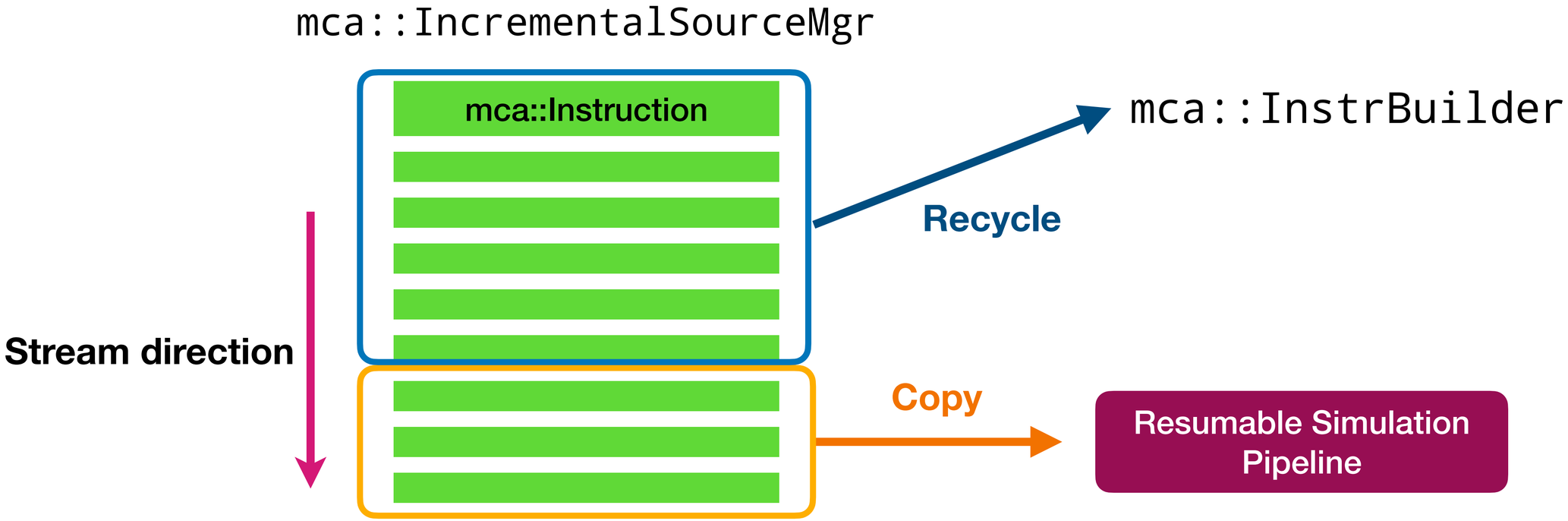
Large memory footprint

Solution: Recycling mca::Instruction



Large memory footprint

Solution: Recycling mca::Instruction



67%

improvement on accumulated memory consumption

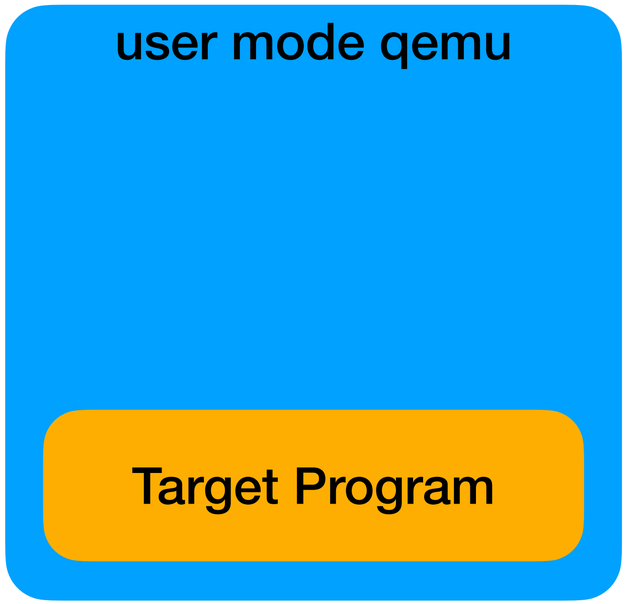
~70%

of the mca::Instruction objects are recycled

Collecting execution traces via QEMU

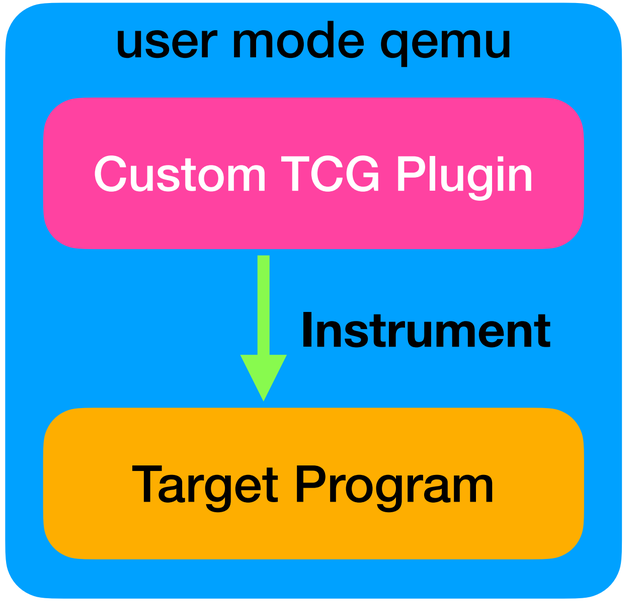
QEMU

MCAD



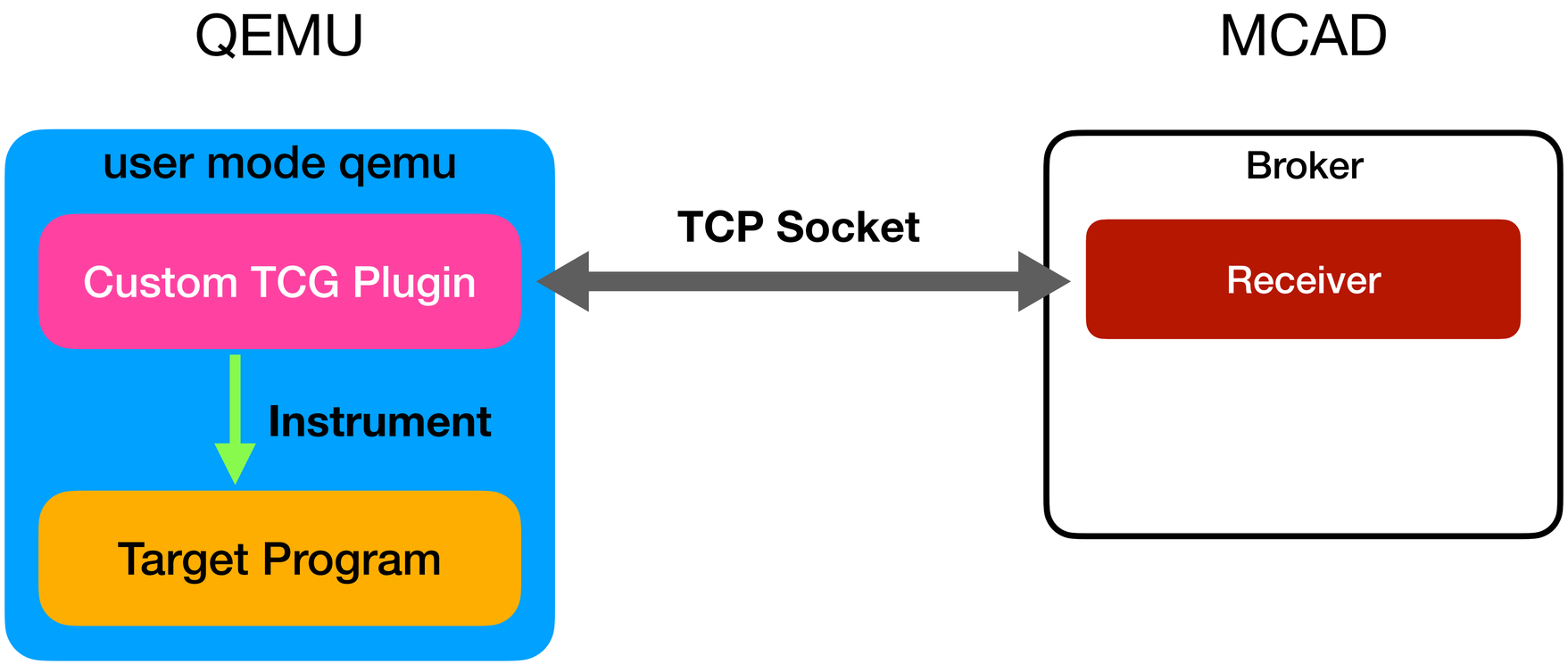
Collecting execution traces via QEMU

QEMU

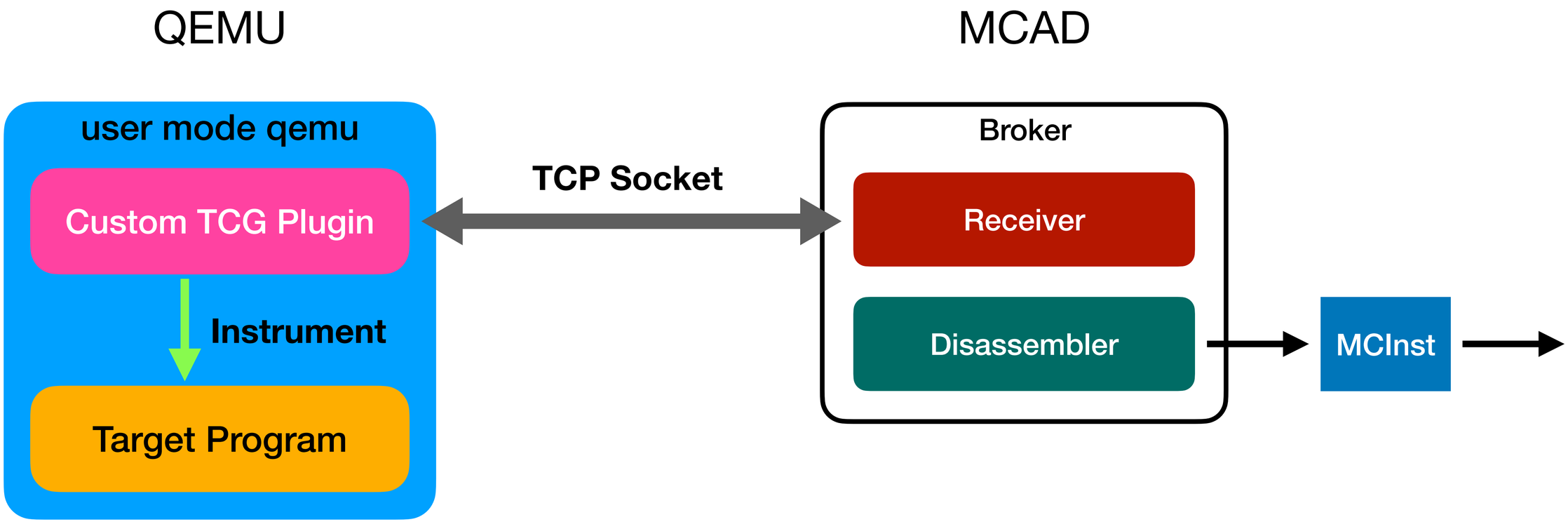


MCAD

Collecting execution traces via QEMU



Collecting execution traces via QEMU



Custom QEMU TCG plugin

Custom QEMU TCG plugin

- The plugin interface allows us to tap into various TCG events to collect executed instructions
  - Example: When a TCG block is translated / executed

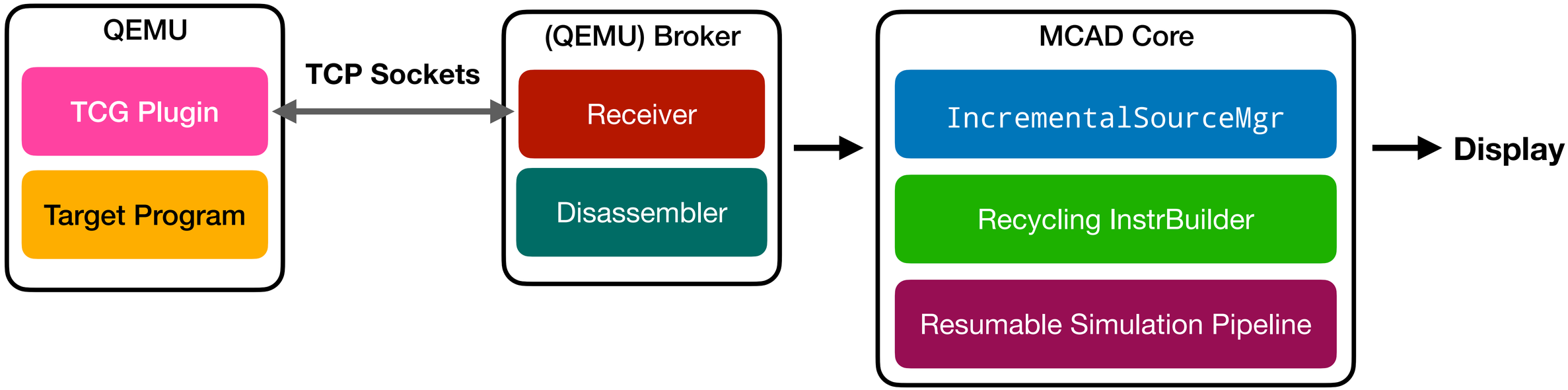
Custom QEMU TCG plugin

- The plugin interface allows us to tap into various TCG events to collect executed instructions
  - Example: When a TCG block is translated / executed
- We also added a few plugin interfaces (not upstreamed yet)
  - Example: Retrieving CPU register values

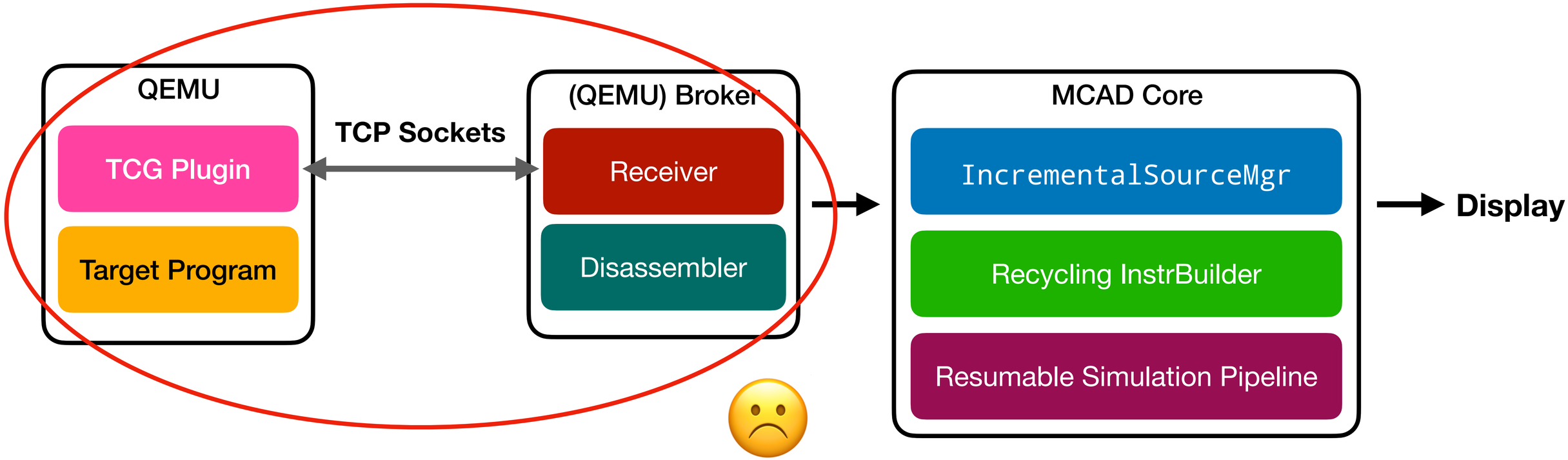
Custom QEMU TCG plugin

- The plugin interface allows us to tap into various TCG events to collect executed instructions
  - Example: When a TCG block is translated / executed
- We also added a few plugin interfaces (not upstreamed yet)
  - Example: Retrieving CPU register values
- Sending raw binary instructions* through TCP sockets

Complete structure of MCAD

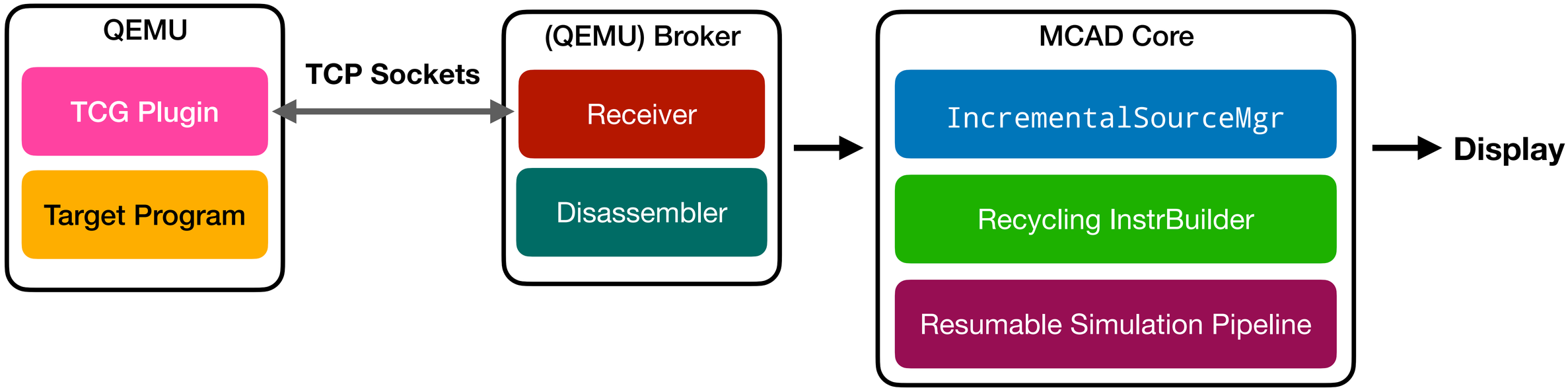


Complete structure of MCAD



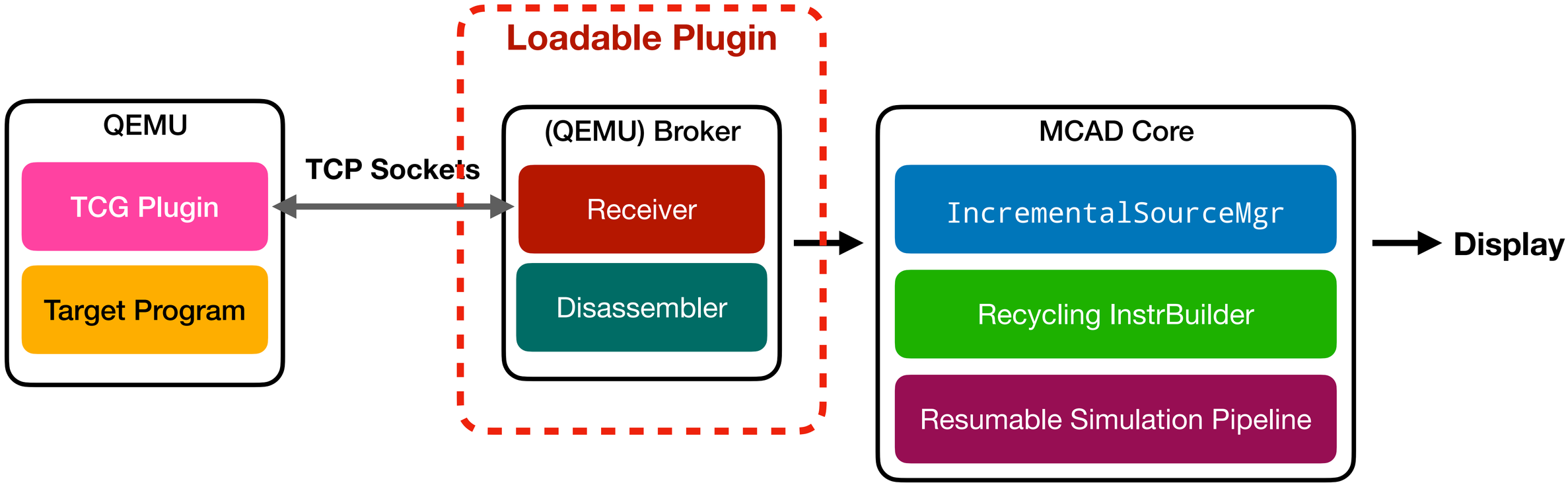
Complete structure of MCAD

A modular design



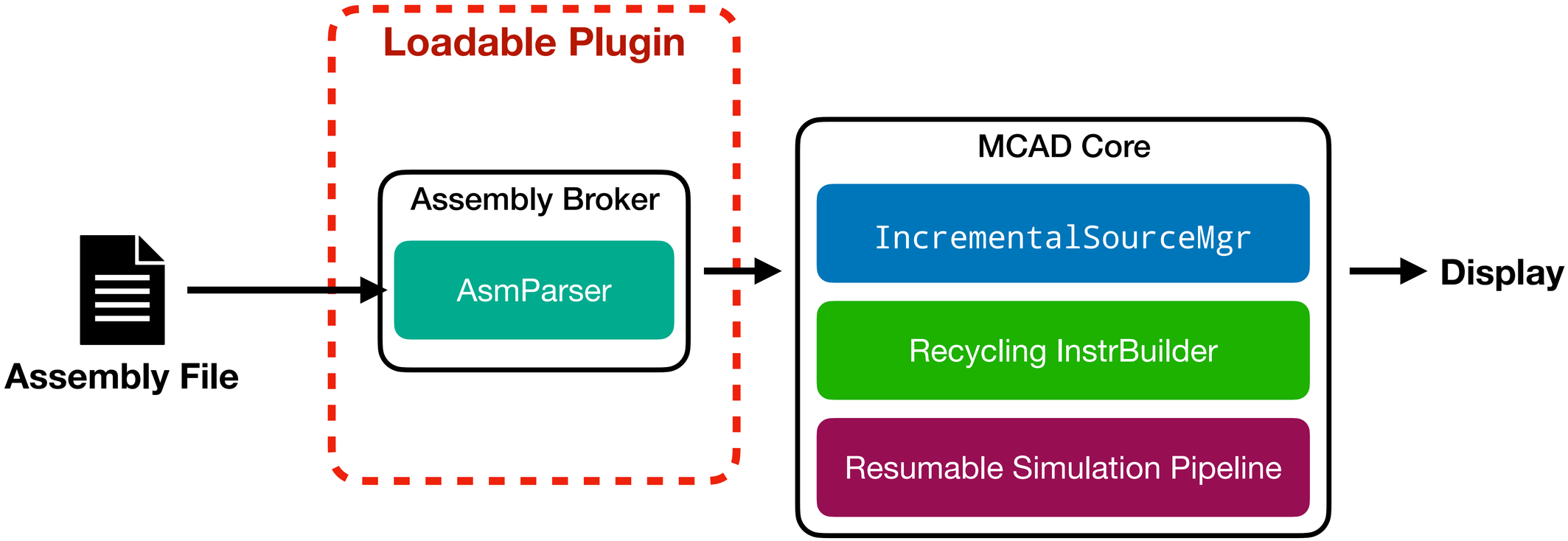
Complete structure of MCAD

A modular design



Complete structure of MCAD

Example: Assembly broker plugin



Evaluation

Scalability compared against llvm-mca

Evaluation

Scalability compared against llvm-mca

- Compare MCAD against llvm-mca (i.e. baseline) on analysis speed and memory consumption

Evaluation

Scalability compared against llvm-mca

- Compare MCAD against llvm-mca (i.e. baseline) on analysis speed and memory consumption
- Using execution trace collected from running **x86_64 FFmpeg 4.2** to decode a 14KB MPEG-4 video file
  - Command: `ffmpeg -i input.mp4 -f null -`
  - Size of the trace: ~27 million x86_64 instructions

Evaluation

Scalability compared against llvm-mca

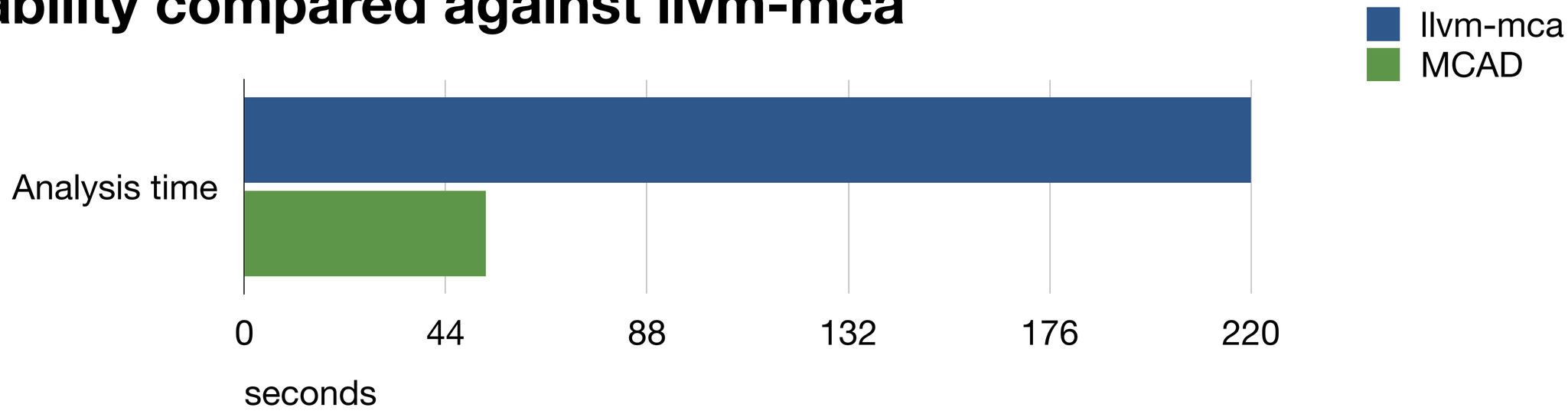
- Compare MCAD against llvm-mca (i.e. baseline) on analysis speed and memory consumption
- Using execution trace collected from running **x86_64 FFmpeg 4.2** to decode a 14KB MPEG-4 video file
  - Command: `ffmpeg -i input.mp4 -f null -`
  - Size of the trace: ~27 million x86_64 instructions
- For baseline, we dump the execution trace (assembly instructions) to a file before feeding into llvm-mca
  - Time measurement on baseline only accounts for llvm-mca's run time. **Excluding** the trace collection time.

Evaluation

Scalability compared against llvm-mca

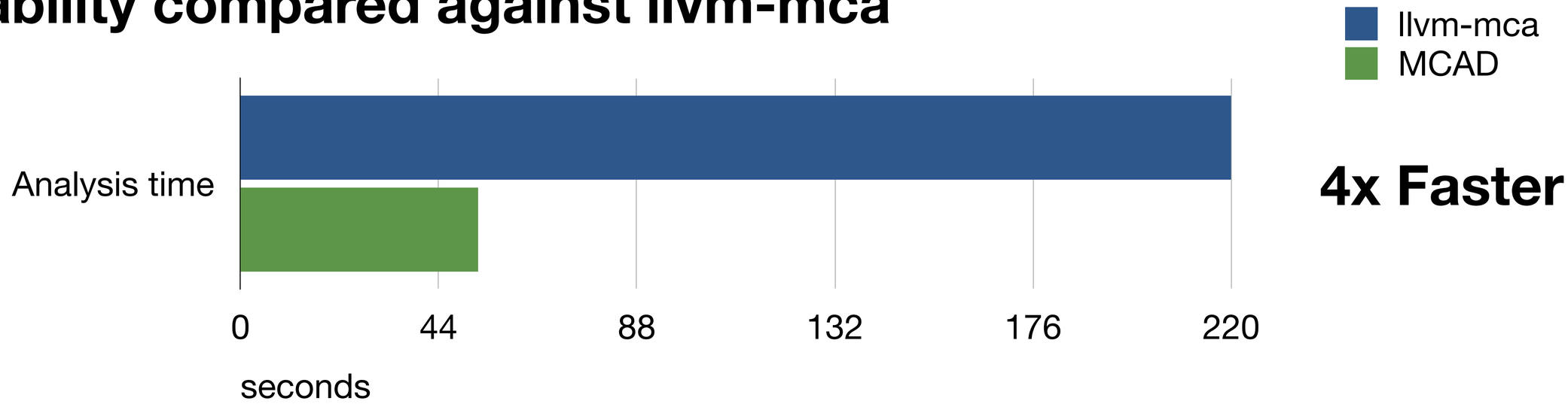
Evaluation

Scalability compared against llvm-mca



Evaluation

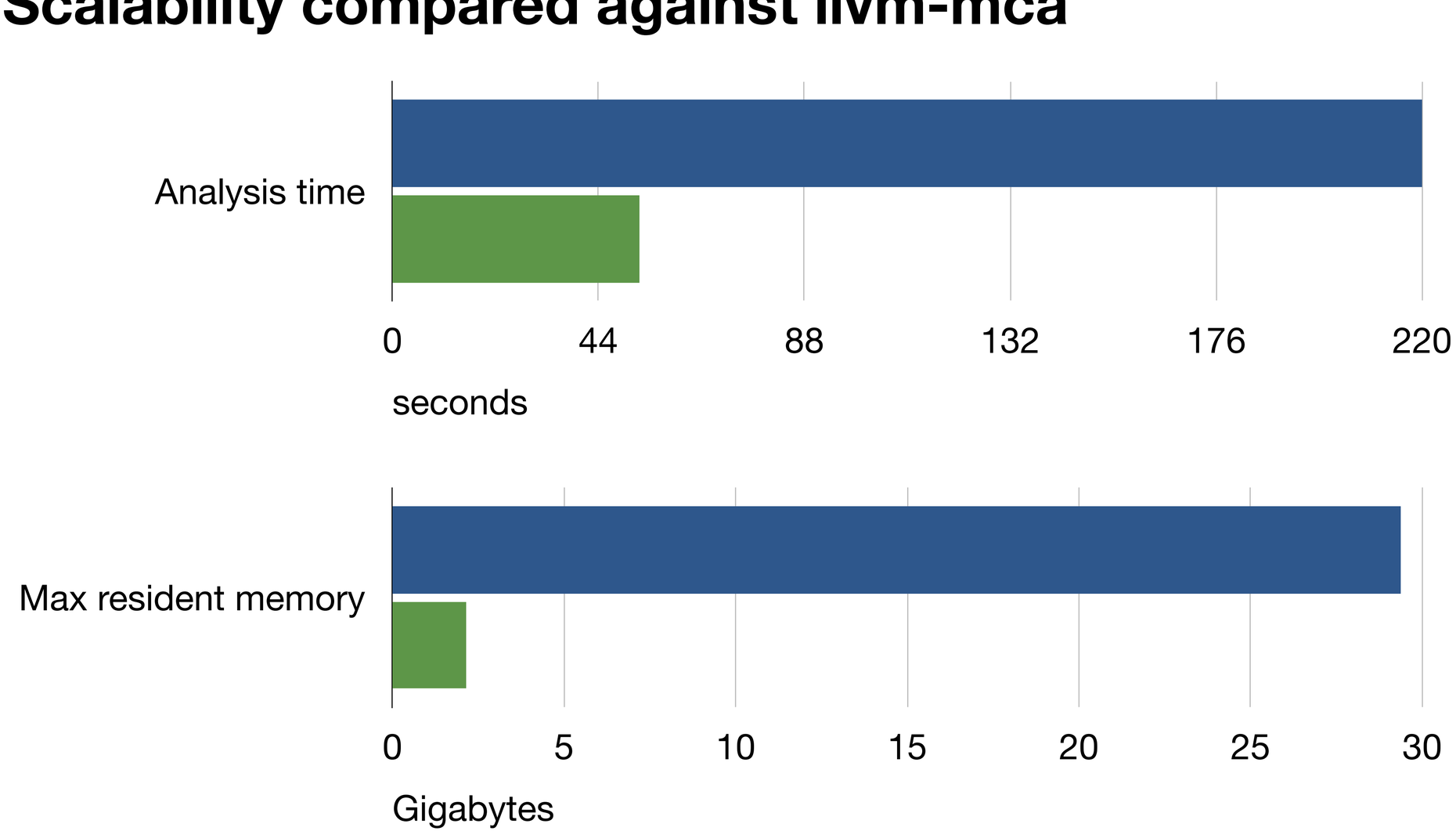
Scalability compared against llvm-mca



4x Faster

Evaluation

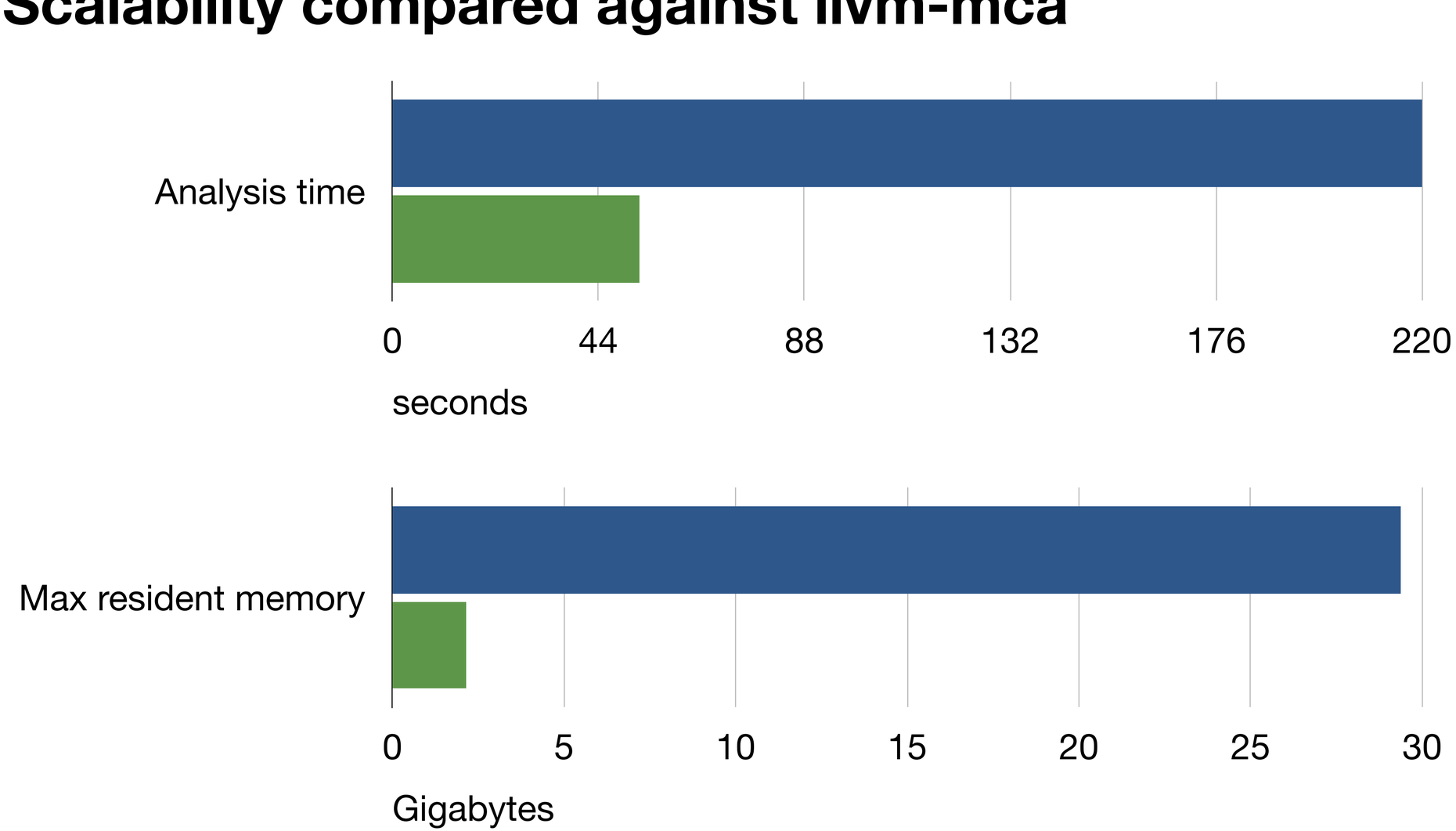
Scalability compared against llvm-mca



4x Faster

Evaluation

Scalability compared against llvm-mca



4x Faster

13x Less

Evaluation

Scalability compared against other static throughput analysis tools

	Analysis Time	Max Resident Memory
uiCA	Timeout after 48h	113 GB
OSACA	Exit w/ error after 24h	N/A
Ithemal	Exit w/ error after 2m	N/A
MCAD	**52.69s**	**2.16 GB**

Outline

Motivation

MCA Daemon (MCAD)

Future Plans

Epilogue

// TODO

// TODO

- More efficient ways to collect traces without QEMU

// TODO

- More efficient ways to collect traces without QEMU
- Analyzing traces from multi-thread programs

// TODO

- More efficient ways to collect traces without QEMU
- Analyzing traces from multi-thread programs
- Improve MCA’s precision via dynamic information (e.g. memory accesses)

// TODO

- More efficient ways to collect traces without QEMU
- Analyzing traces from multi-thread programs
- Improve MCA’s precision via dynamic information (e.g. memory accesses)
- Visualizing analysis results. Or: Improve MCA’s result display

// TODO

- More efficient ways to collect traces without QEMU
- Analyzing traces from multi-thread programs
- Improve MCA's precision via dynamic information (e.g. memory accesses)
- Visualizing analysis results. Or: Improve MCA's result display
  - Example: *Loadable plugins* for custom display of the result

// TODO

- More efficient ways to collect traces without QEMU
- Analyzing traces from multi-thread programs
- Improve MCA's precision via dynamic information (e.g. memory accesses)
- Visualizing analysis results. Or: Improve MCA's result display
  - Example: *Loadable plugins* for custom display of the result
- Going upstream: QEMU & LLVM

Going upstream: LLVM

The plan

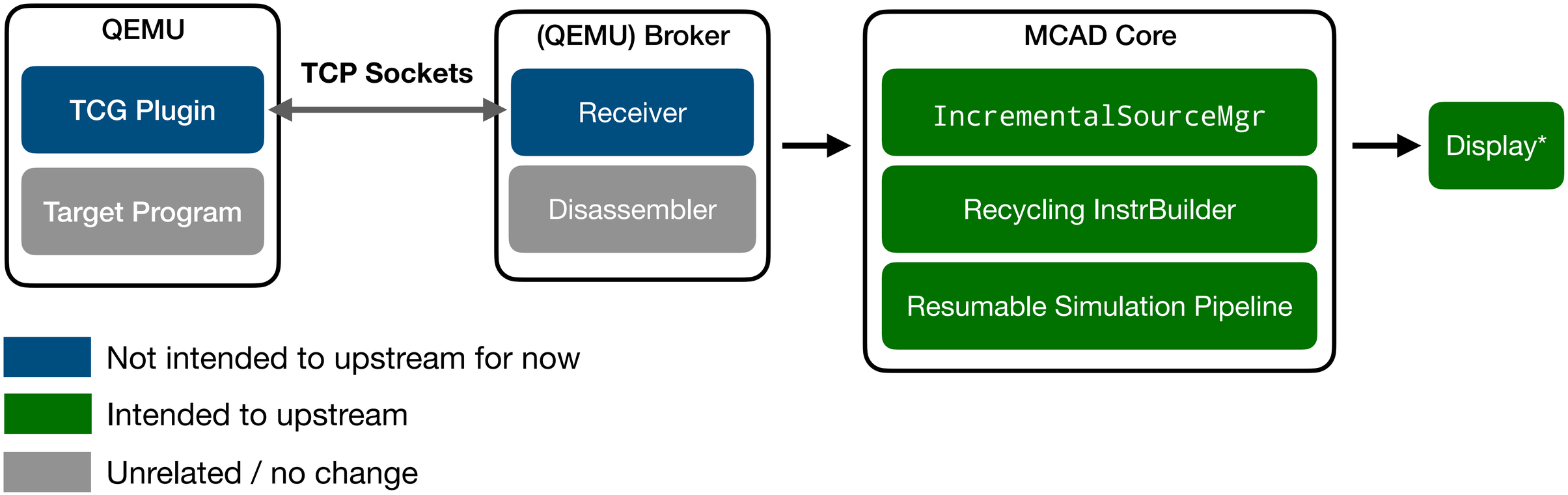
Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first

Going upstream: LLVM

The plan: components to upstream



Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first

Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first
- QEMU broker plugin & our TCG plugin will be maintained out-of-tree

Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first
- QEMU broker plugin & our TCG plugin will be maintained out-of-tree
- We’re not sure about upstreaming rest of the tool right now

Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first
- QEMU broker plugin & our TCG plugin will be maintained out-of-tree
- We're not sure about upstreaming rest of the tool right now
  - With the assembly broker, MCAD can be a drop-in replacement for llvm-mca…with even more features (e.g. the broker plugin infrastructure)

Going upstream: LLVM

The plan

- We would like to upstream components that are beneficial to the **core MCA libraries** first
- QEMU broker plugin & our TCG plugin will be maintained out-of-tree
- We're not sure about upstreaming rest of the tool right now
  - With the assembly broker, MCAD can be a drop-in replacement for llvm-mca…with even more features (e.g. the broker plugin infrastructure)
  - Some of the (advanced) interfaces in broker plugin are only used by QEMU broker. So, without the latter, it's not well tested.

Outline

Motivation

MCA Daemon (MCAD)

Future Plans

Epilogue

Summary

Summary

- **MCA Daemon (MCAD)** is a high-performance *hybrid* throughput analysis tool built on top of LLVM MCA libraries

Summary

- **MCA Daemon (MCAD)** is a high-performance *hybrid* throughput analysis tool built on top of LLVM MCA libraries
  - Online, whole-program analysis on real-world applications

Summary

- **MCA Daemon (MCAD)** is a high-performance *hybrid* throughput analysis tool built on top of LLVM MCA libraries
  - Online, whole-program analysis on real-world applications
  - Scale up with large-scale programs with tens of **millions** of instructions

Summary

- **MCA Daemon (MCAD)** is a high-performance *hybrid* throughput analysis tool built on top of LLVM MCA libraries
  - Online, whole-program analysis on real-world applications
  - Scale up with large-scale programs with tens of **millions** of instructions
- We improved the performance & flexibility of core MCA libraries

Summary

- **MCA Daemon (MCAD)** is a high-performance *hybrid* throughput analysis tool built on top of LLVM MCA libraries
  - Online, whole-program analysis on real-world applications
  - Scale up with large-scale programs with tens of **millions** of instructions
- We improved the performance & flexibility of core MCA libraries
  - We would like to merge these changes upstream to benefit the wider community

Source code

https://github.com/securesystemslab/LLVM-MCA-Daemon

Acknowledgements

This material is based upon work partially supported by the Defense Advanced Research Projects Agency (DARPA) under contract N66001-20-C-4027

Special Thanks:

Galois Inc. (https://galois.com) **Immunant Inc.** (https://immunant.com)


Thank You!
Q&A
Me
Aldrich Park @ UC Irvine, California

Appendix

Introduction to MCA

An example

test/tools/llvm-mca/X86/BtVer2/dot-product.s

```
vmulps    %xmm0, %xmm1, %xmm2
vhaddps   %xmm2, %xmm2, %xmm3
vhaddps   %xmm3, %xmm3, %xmm4
```

```
llvm-mca -mtriple=x86_64 -mcpu=btver2 \
         -iterations=300 dot-products.s
```

Timeline

```
                    012345
Index     0123456789

[0,0]     DeeER.    .    .   vmulps
[0,1]     D==eeeER  .    .   vhaddps
[0,2]     .D====eeeER    .   vhaddps
[1,0]     .DeeE-----R    .   vmulps
[1,1]     . D=eeeE---R   .   vhaddps
[1,2]     . D====eeeER   .   vhaddps
[2,0]     .   DeeE-----R .   vmulps
[2,1]     .   D====eeeER .   vhaddps
[2,2]     .    D======eeeER   vhaddps
```

Introduction to MCA

An example

test/tools/llvm-mca/X86/BtVer2/dot-product.s

```
vmulps    %xmm0, %xmm1, %xmm2
vhaddps   %xmm2, %xmm2, %xmm3
vhaddps   %xmm3, %xmm3, %xmm4
```

```
llvm-mca -mtriple=x86_64 -mcpu=btver2 \
         -iterations=300 dot-products.s
```

Timeline

```
                    012345
Index     0123456789

[0,0]     DeeER.    .    .   vmulps
[0,1]     D==eeeER  .    .   vhaddps
[0,2]     .D====eeeER   .   vhaddps
[1,0]     .DeeE-----R   .   vmulps
[1,1]     . D=eeeE---R  .   vhaddps
[1,2]     . D====eeeER  .   vhaddps
[2,0]     .   DeeE-----R .  vmulps
[2,1]     .   D====eeeER .  vhaddps
[2,2]     .    D======eeeER vhaddps
```