Warm Starts in Junction

Or, snapshotting and restoring a kernel-bypass library OS

6.5810 | Baltasar Dinis | Ben Holmes

Roadmap

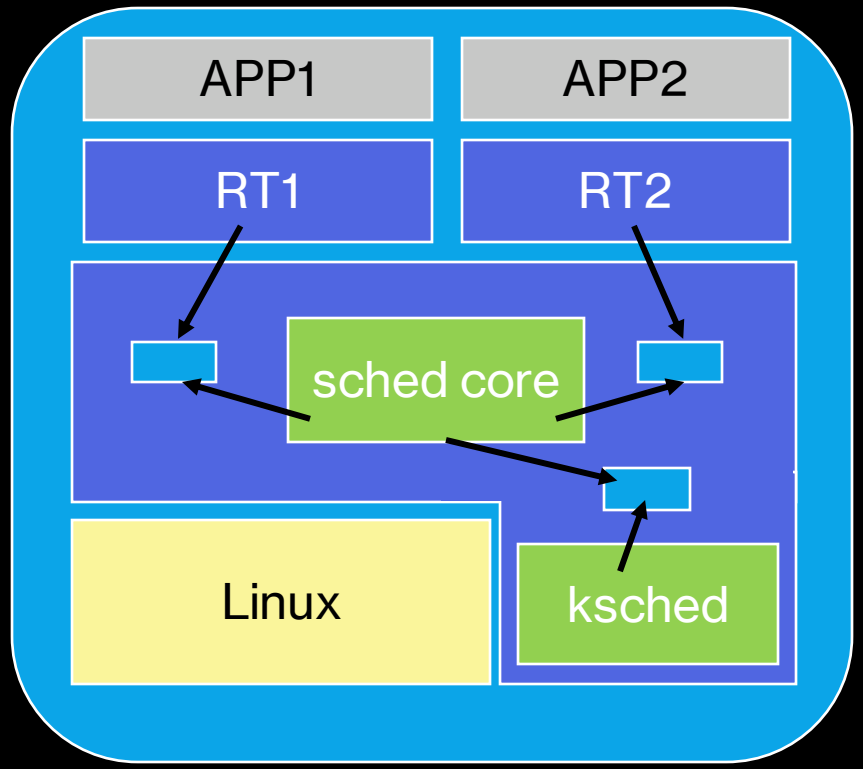
- **Backspace:** what is this Junction you keep talking about?
- What would we need to even snapshot?
- How to snapshot?
- Is there any speedup?
- Where do we go from here?

Motivation: we want single millisecond-scale serverless

Requirements for Serverless	Done?
Scalable	✕
Isolation	✕
Fast	✕
Handles Burstiness	✕
Familiar API	✕
Starts fast	✕

Caladan (OSDI'20) **Overview**

- **Goal:** reallocate cores at the microsecond timescale
  - This enables the system to **handle burstiness** efficiently
- Busy poll for control signals
- Shared memory regions allow for asynchronous scheduling

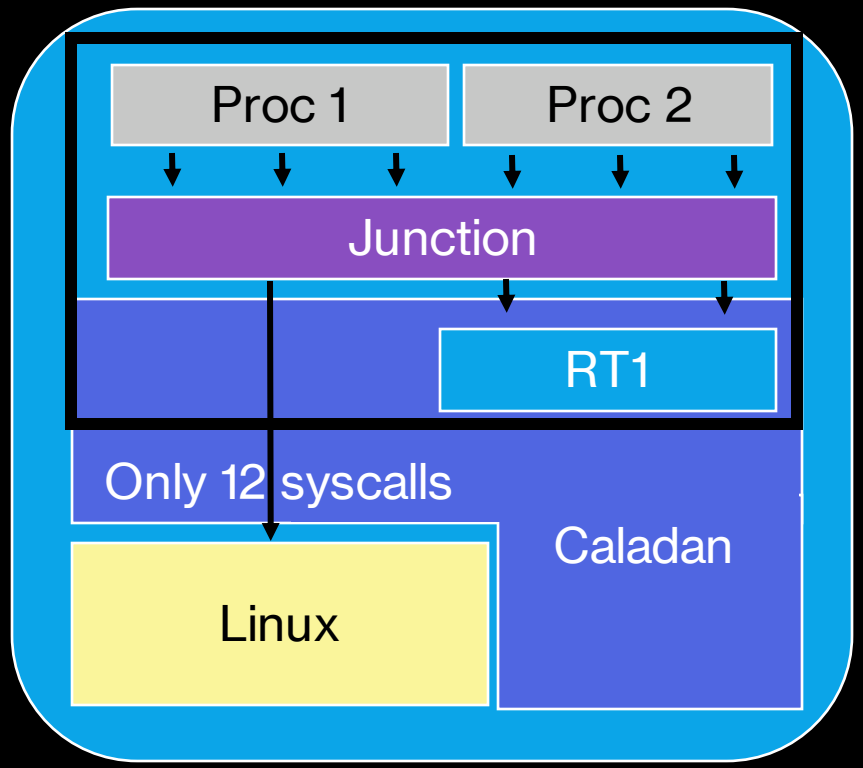


Motivation: we want single millisecond-scale serverless

Requirements for Serverless	Done?
Scalable	✓
Isolation	✗
Fast	✓
Handles Burstiness	✓
Familiar API	✗
Starts fast	✗

Junction's Techniques

- **Scalability:** Wrap around Caladan and use modern NIC features
- **Isolation:**
  - Implement the majority of Linux syscalls in user-space, `seccomp` away the others
  - Use `cgroups` and `chroot`
- **Compatibility:** Expose the same API as Linux to run unmodified binaries
  - Provide a modified `libc` to avoid going through the `seccomp` filter every time.



Junction has potential as a serverless substrate

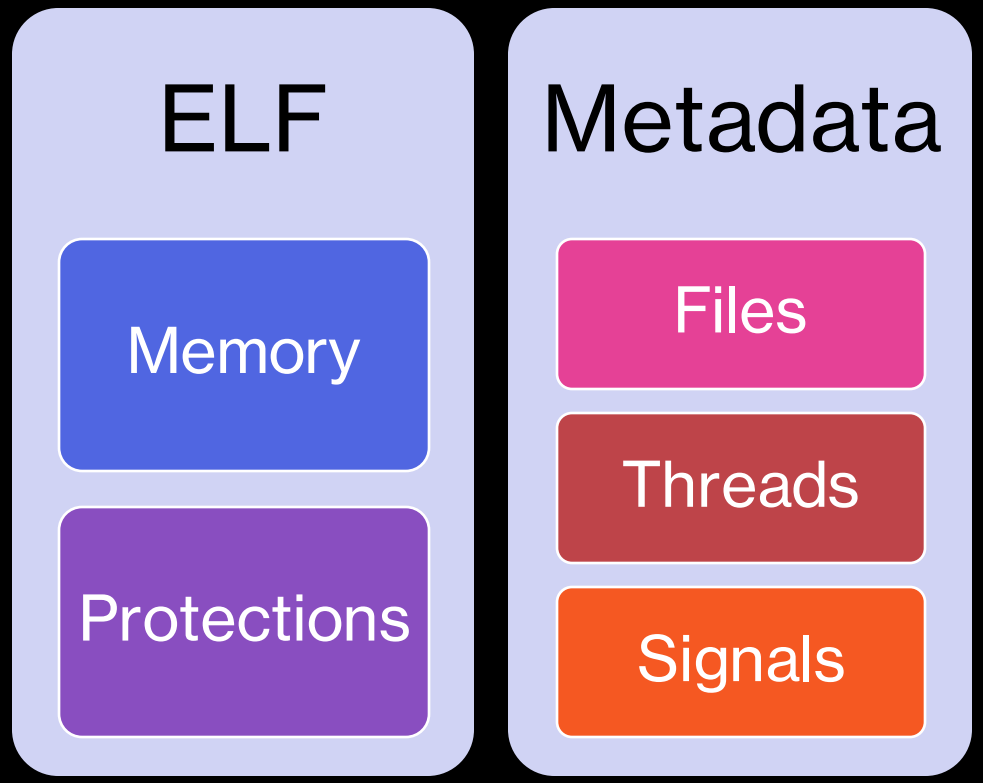
Requirements for Serverless	Done?
Scalable	✓
Isolation	✓
Fast	✓
Handles Burstiness	✓
Familiar API	✓
Starts fast	✗

Roadmap

- Backspace: what is this Junction you keep talking about?
- **What** would we need to even snapshot?
- How to snapshot?
- Is there any speedup?
- Where do we go from here?

The components of a snapshot

- The current memory of the process
- The registers in use by the threads
- Signal Queues
- The file table



Roadmap

- Backspace: what is this Junction you keep talking about?
- What would we need to even snapshot?
- **How** to snapshot?
- Is there any speedup?
- Where do we go from here?

How to snapshot
Step 1: you need a way in

- Can we just add a new function/system call?
  - What about unmodified binaries?
- Can Junction do it at a particular time?
- What about a signal?
  - Can we reuse one of the well-established signals?
  - Can we create a new signal?

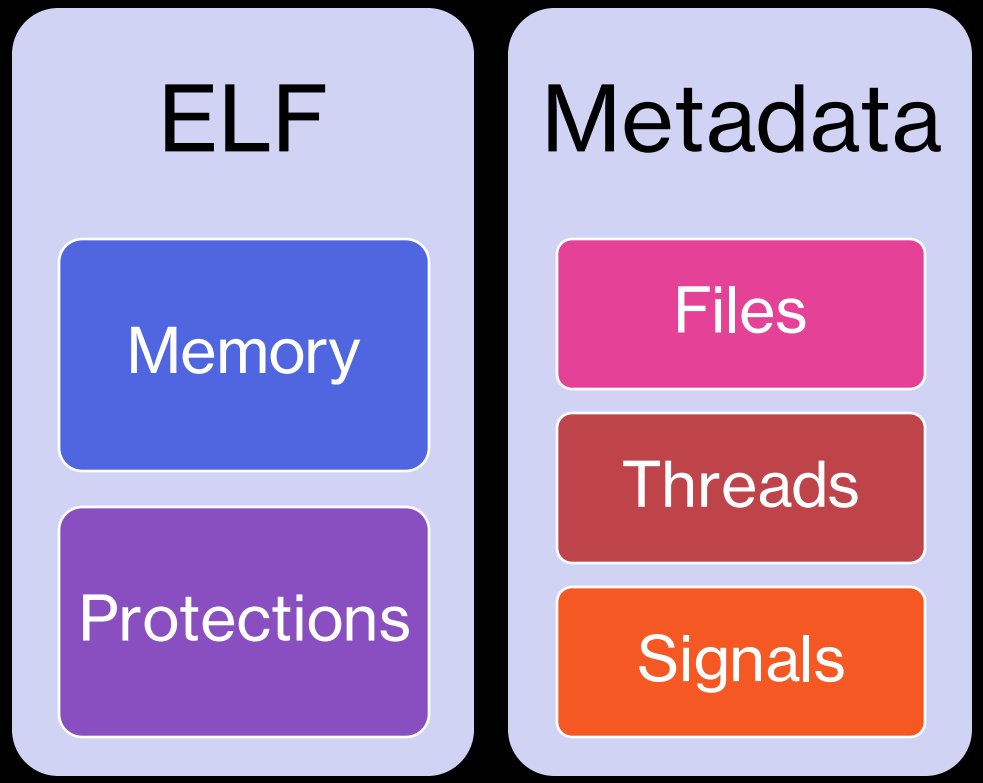
```
int snapshot(
     char const * metadata_filename,
     char const * elf_filename)

 0 --> returning from a snapshot
 1 --> returning from a restore
-1 --> error
```

How to snapshot
Step 2: everybody stop

- We need to make sure nothing is running to **snapshot consistently**
- The calling thread is running the snapshot entrypoint code
  - What about the other threads?

How to snapshot
Step 3: take a picture

- Go over Junction's internal structures and write the snapshot
  - The memory and its protections go into an ELF file, which **can be loaded** like any other
  - The metadata goes into a separate file, in a **serialized format**



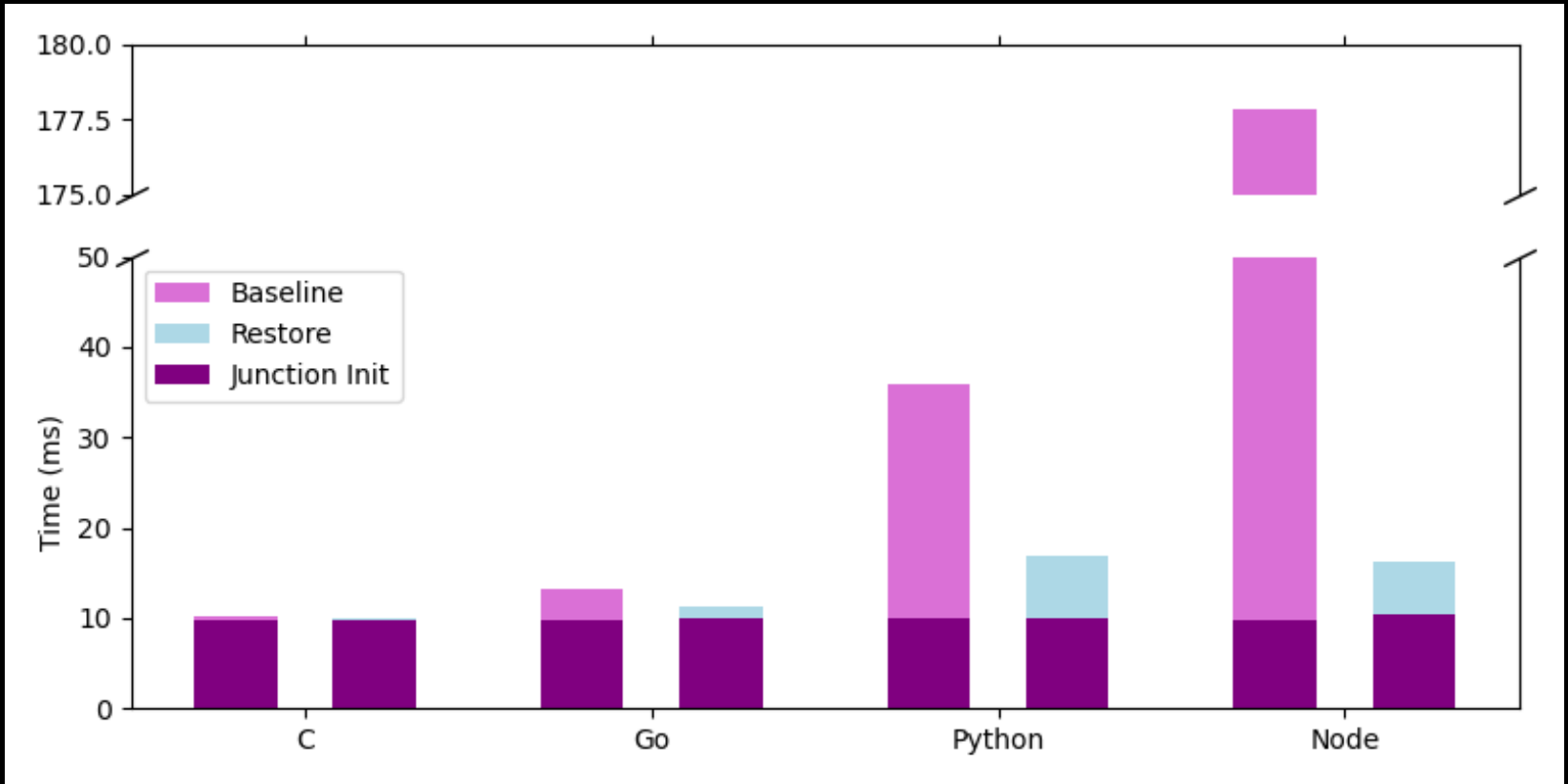
How to restore

- **Step 1:** restore Junction's internal structures
  - From the serialized metadata
- **Step 2:** load the ELF file to restore the memory mappings
- **Step 3:** restore the registers
- **Step 4:** return from the system call
- **Step 5:** Profit **$$$**

Roadmap

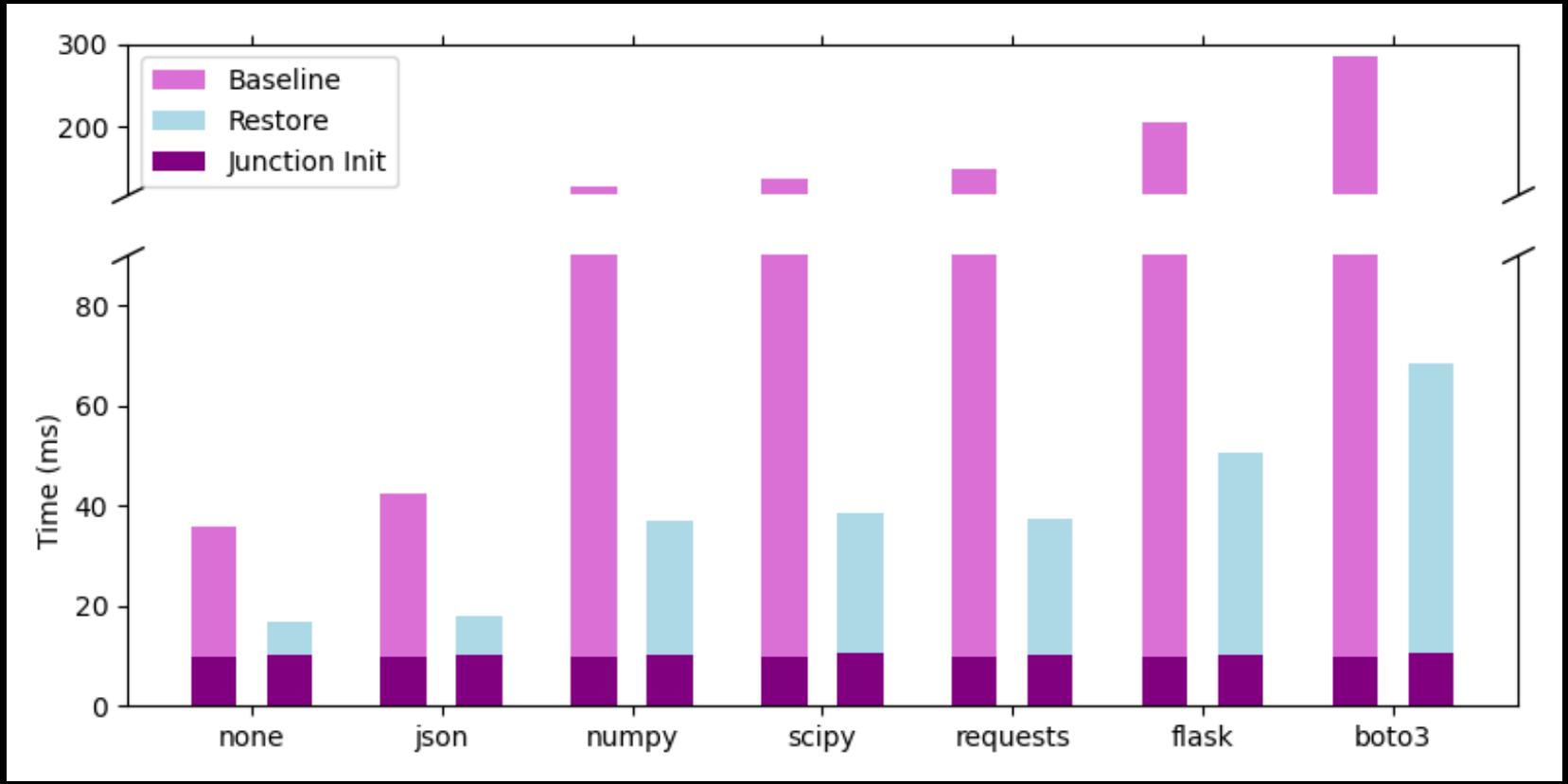
- Backspace: what is this Junction you keep talking about?
- What would we need to even snapshot?
- How to snapshot?
- Is there any **speedup**?
- Where do we go from here?

Hello, World!



For interpreted/JITed languages we can get big savings

Common Serverless Dependencies



Good savings, but we could do better with more optimizations

Roadmap

- `Backspace`: what is this Junction you keep talking about?
- What would we need to even snapshot?
- How to snapshot?
- Is there any speedup?
- **Where** do we go from here?

Where do we go from here?

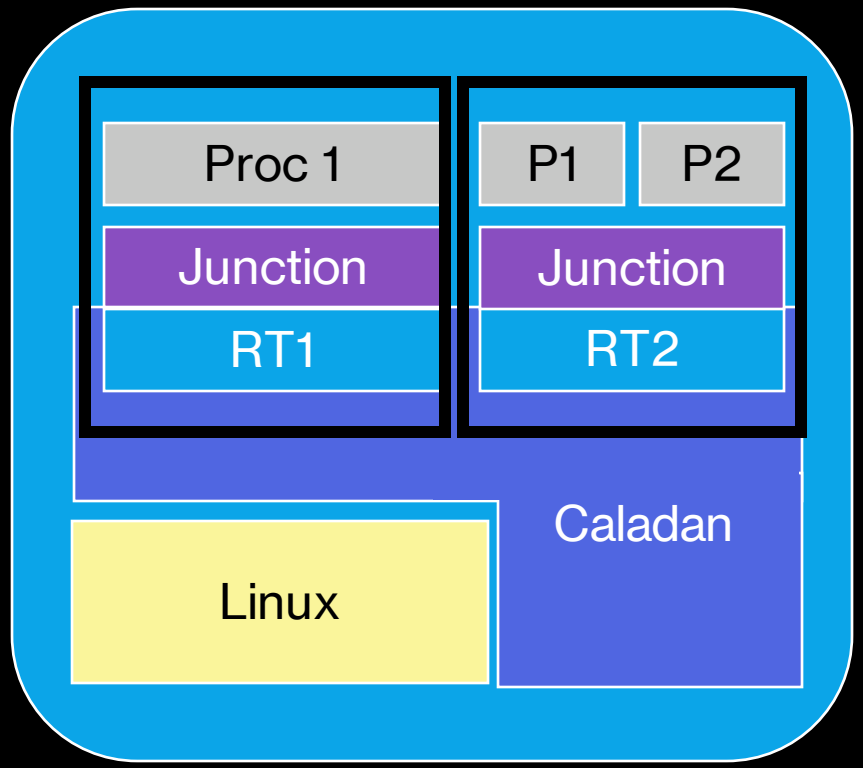
- Sub-millisecond start
  - I.e., make cold starts hot
- But how?
  - Handle 0-pages
  - Compress the snapshot and use hardware decompression
  - Pre-warm the instances
    - What about provisioned concurrency?
    - Stem-cell snapshots, instances can have diffs applied to them

Questions

Thank you

Junction Makes Kernel-Bypass Practical

- Existing kernel bypass systems can't scale and lack isolation
- Junction provides kernel bypass for unmodified applications with scalability and strong isolation



Junction's Techniques

- Wraps around Caladan to provide kernel bypass core scheduling for unmodified applications
- Reimplements the Linux system call interface to expose only about a dozen syscalls
- Lightweight threading that avoids syscalls
- Leverages modern NIC features to reduce pinned memory

Junction Makes Kernel-Bypass Practical

- Bypassing the kernel significantly reduces tail latency
- Existing kernel bypass systems:
  - No **scalability**
  - No **isolation**
  - Can't run **unmodified binaries**
- Junction addresses all of these issues