

Remote Direct Code Execution

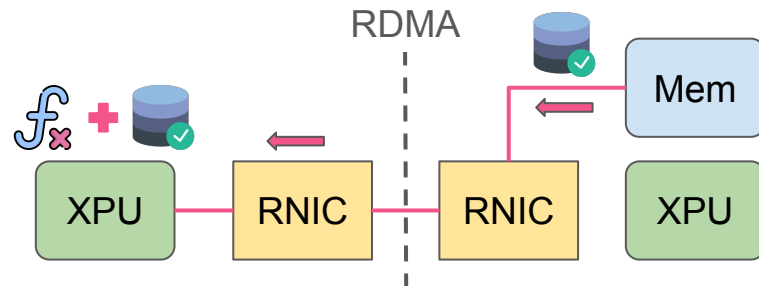
Yibo Huang, Yiming Qiu, Daqian Ding, Patrick Tser Jern Kon, Yiwen Zhang, Yuzhou Mao,
Archit Bhatnagar, Mosharaf Chowdhury, Srinivas Devadas, Jiarong Xing, Ang Chen



RDMA is already the workhorse for data access

- Remote Direct Memory Access (RDMA):

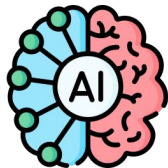
- Hardware offloading
- Low latency, high throughput
- Minimal overhead



- Widely deployed for high-performance datacenter workloads



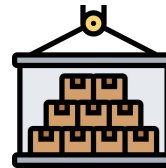
Key-value database



AI/ML training



HPC

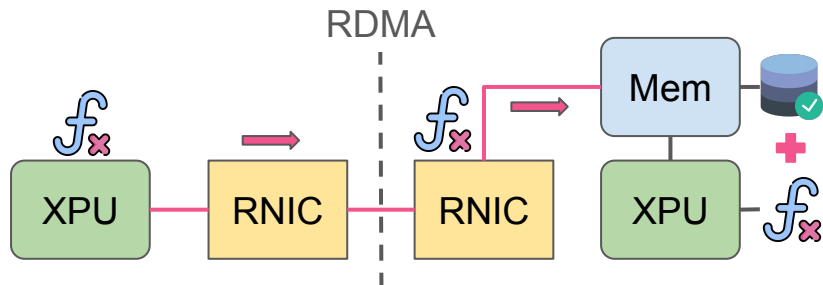


Storage

.....

RDX: Elevating Remote Direct Memory Access to code eXecution

- Inject and execute code on remote node via RDMA
- Intuition: Code is “data”
 - Inject a compiled binary to remote executable memory
 - Attach the code to software entities for execution, e.g., sandbox/hooks
 - Negligible overhead, transparent to RDMA stack and remote node



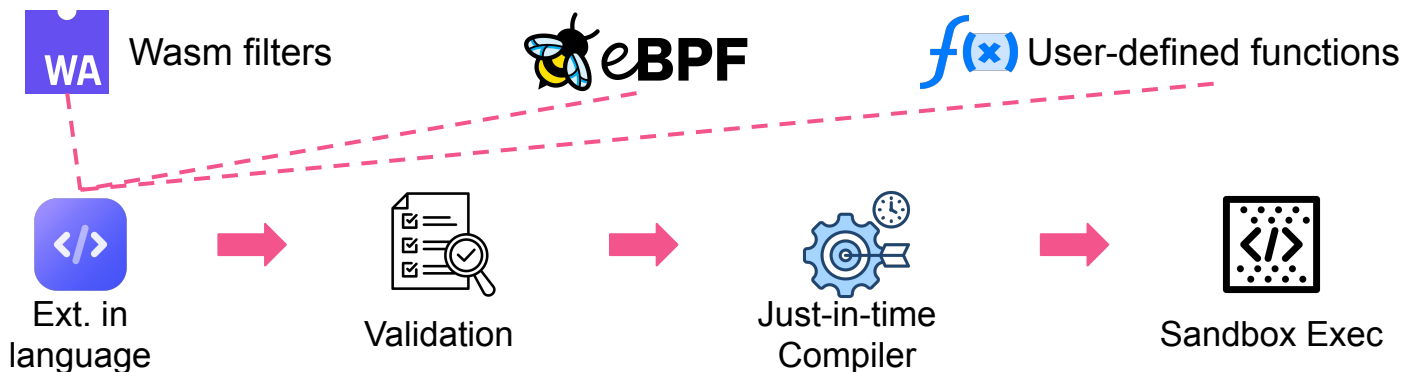
Ideally, enable the infrastructure itself as a runtime-programmable computer in a **high-velocity manner**

...

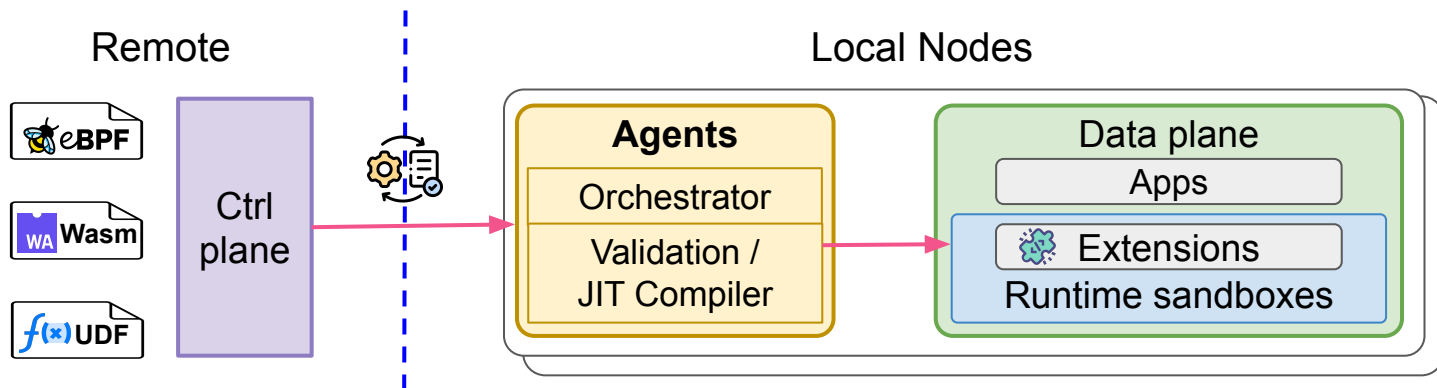
Velocity
Manageability
Availability

Why RDX? A case of executing runtime extensions (SOTA)

- Runtime extensions
 - Express/Execute custom logics for runtime workload dynamics
 - Loaded/Unloaded on the fly without application or service downtime
- Used for security defenses, performance acceleration, monitoring/telemetry

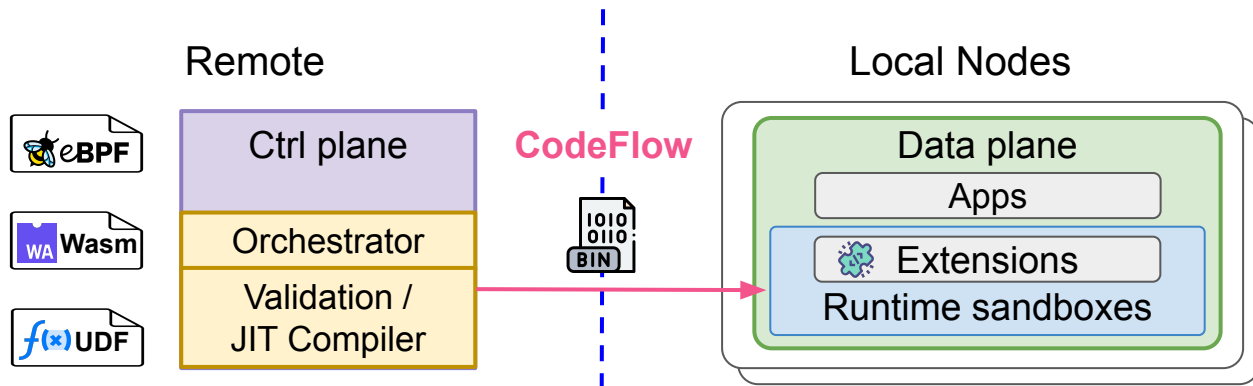


State of the art runtime extension frameworks & limitations



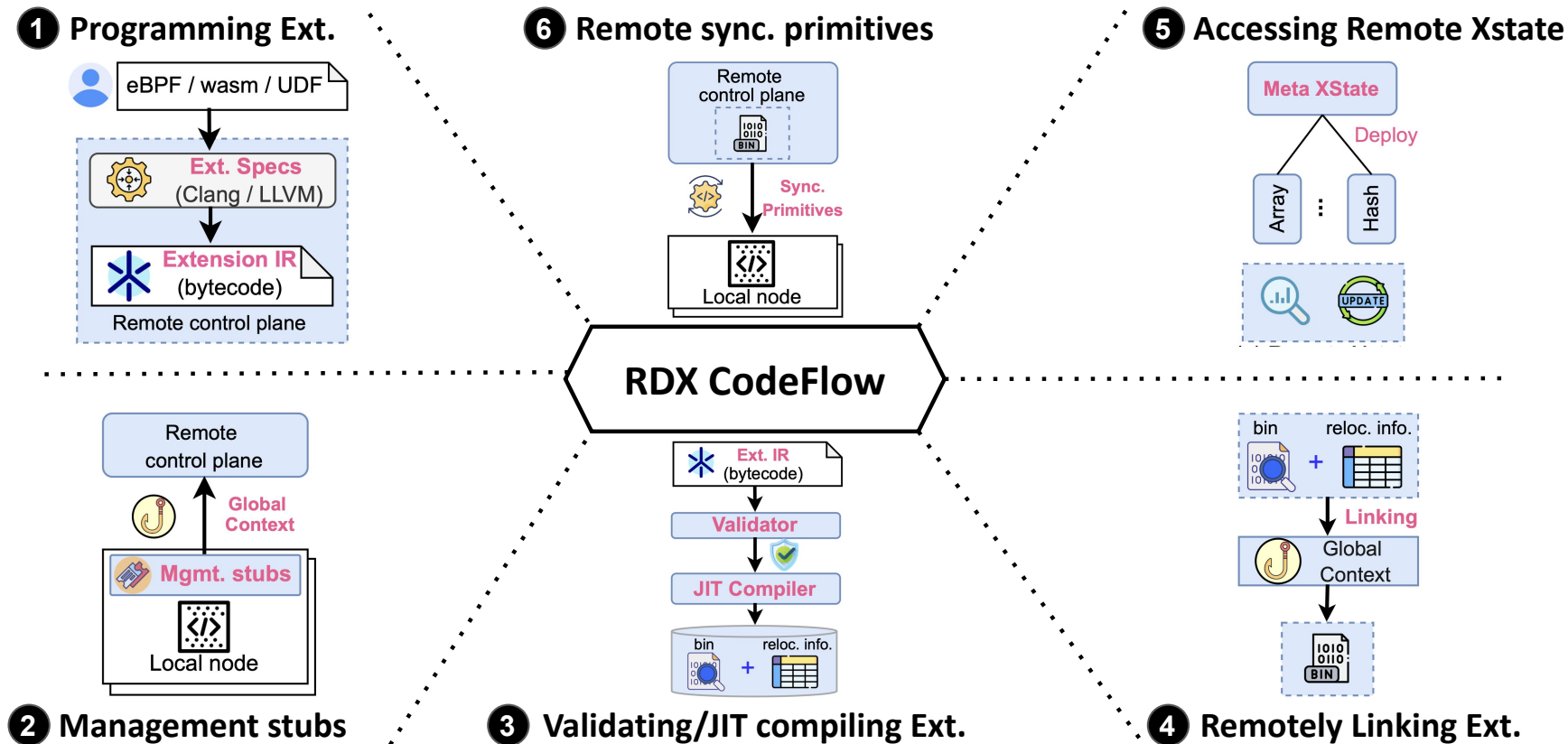
- Agent-based architecture to enforce extensions at scale
 - Deployed per pod in Kubernetes based containerized clouds
- However, agent causes high injection delay, mismatching with us-scale workloads
- Updating interdependent extensions introduce inconsistency to the data path
- The shared hardware resource contention between control and data path
 - High density agents + frequent ext. state access

RDX in a Nutshell: An Agentless Architecture



- An agentless architecture over RDX with a centralized remote controller
- Decouple control path ext. injection from data path ext. execution
- Remote direct runtime extension (re)programming end-to-end
- Ultra-fast injection (*us*-scale), consistency guarantees, minimized contention

RDX building blocks: an end-to-end view

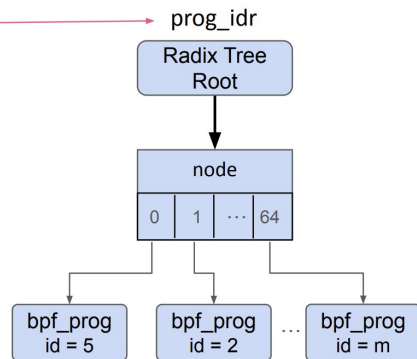


Planting remote management stubs

eBPF program as an example

>= 30 variables

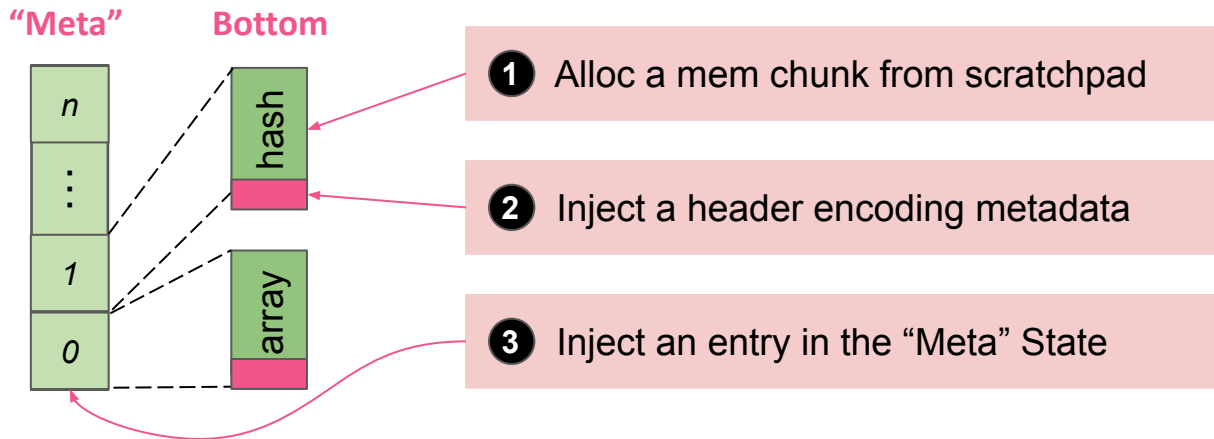
```
/* bpf/syscall.c */
static DEFINE_IDR(prog_idr);
/* linux/filter.h */
struct bpf_prog {
    int id;
    int jited_len;
    void *bpf_func;
    struct bpf_insn insnsi[0];
    struct bpf_prog_aux {
        struct bpf_prog_stats *stats;
        ...
    } *aux; ...
};
```



- Expose local sandbox's runtime context to remote control plane when sandbox boots
 - **ctx_init**: create sandbox context by preloading empty extensions
 - **ctx_register**: register memory address for control knobs, e.g., select metadata
 - **ctx_teardown**: assist in detaching extensions based on reference counting

Accessing remote extension states

e.g., eBPF array and hash map



- Keep extension state in various data structures called **State**
- Challenge: Dynamic lifecycle of XState due to various XState types and sizes
- Instantiate a "Meta" XState with two-layer design at setup time
 - Create an XState instance by indexing it into "Meta" XState at runtime
 - Update and lookup XState data elements from remote control plane and local ext.

Open question: Can we trust RDX?

What if someone sees extension states they shouldn't?



A remote gatekeeper with role-based privilege model

Confidentiality

What if someone tampers with code or states?



Runtime guards + Runtime introspection w/ checksumming

Integrity

RDX Trust Plane

Availability

What if an RDX extension has faulty or overruns its budget?

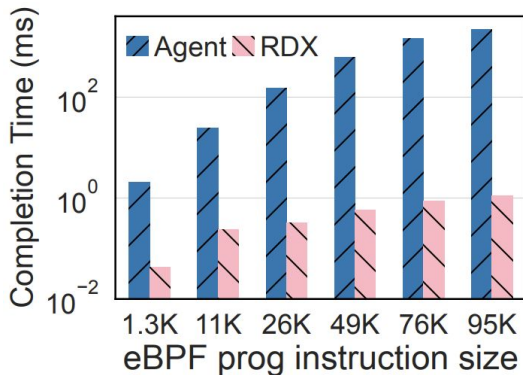


Runtime budget limits + ext. state machine + quick ext. rollbacks

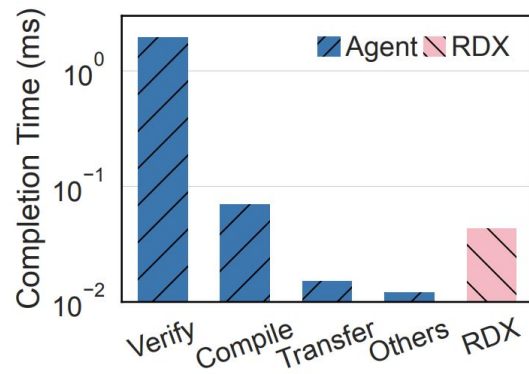
Toward RDX

Prototyped RDX with eBPF

- Socket Filter/Kprobe/XDP
- **20+** eBPF programs from Linux upstream/Cilium
- Reduce injection time by **47~1982X**



(a) Prog. load overhead.



(b) Time breakdown (1.3K).



RDMA

Static logics at compile time
Data semantic



RDX

High-velocity logics at runtime
Code semantic

- A new modality of “Remote Direct” primitives!