Methods to reduce cold-start time

Normal execution

- GPUs spatially partitioned -> multiple functions
- Each function associated with own runtime (CUDA context)
    - Can be done with Nvidia MPS
    - Needs to be initialized -> high cold start latency, memory footprint, etc
- Usually fixed-size containers -> not ideal memory usage

Streambox

- For one inference workflow: use only one runtime (not multiple for each subtask)
  - How: Use GPU streams (provided e.g. by CUDA) -> shared address space for concurrent executions
- Challenges
  - C1: Efficient memory allocation and sharing (coarse-grained memory, no auto-scaling)
  - C2: Inter-function communication (redundant data transfers)
  - C3: High-performance parallelism (shared PCIe link bottleneck)

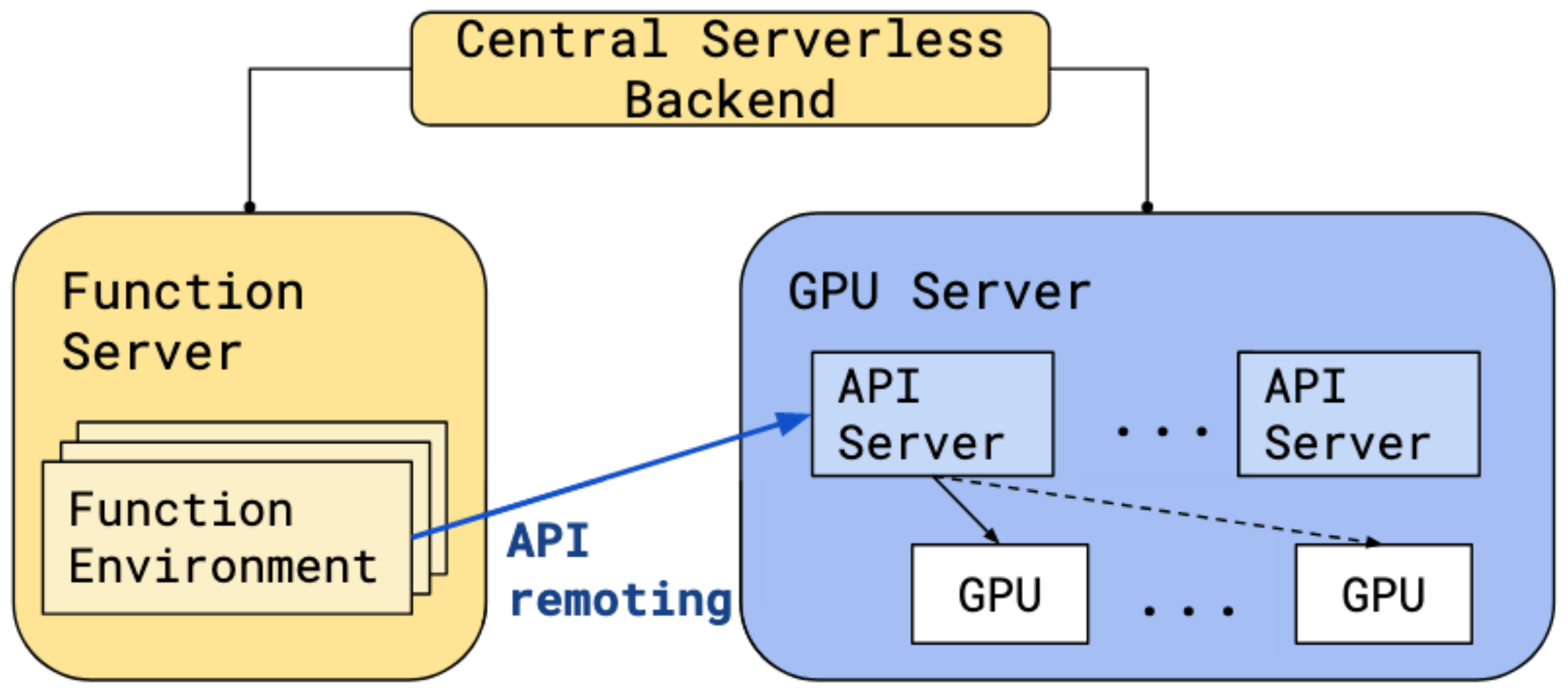
Streambox

- Solutions
  - C1: Efficient memory allocation and sharing -> allocate and recycle memory at layer granularity
  - C2: Inter-stream communication -> unified communication framework, store intermediate data in GPU memory, elastic communication store
  - C3: High-performance parallelism -> fine-grained PCIe bandwidth sharing (by partitioning data of functions in blocks)

DSGF

- Idea: independently manage and scale CPU and GPU resources
- Challenges:
  - C1: Preserving the serverless programming model: remote GPU should appear local. Requesting and utilizing a GPU should not require any management or knowledge of its location.
  - C2: Preserving the performance promise of GPU acceleration in the face of overheads introduced by remote execution.
  - C3: Balancing load and maximizing utilization of remote GPUs.

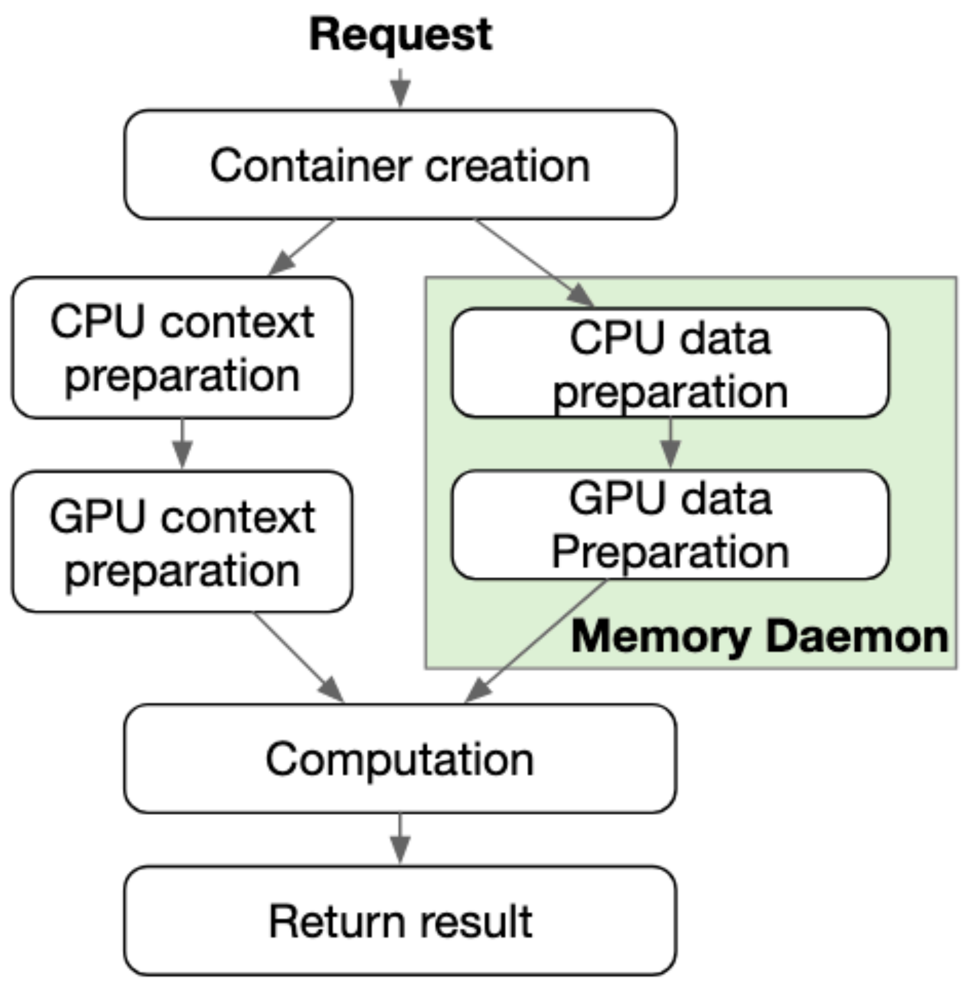
DSGF

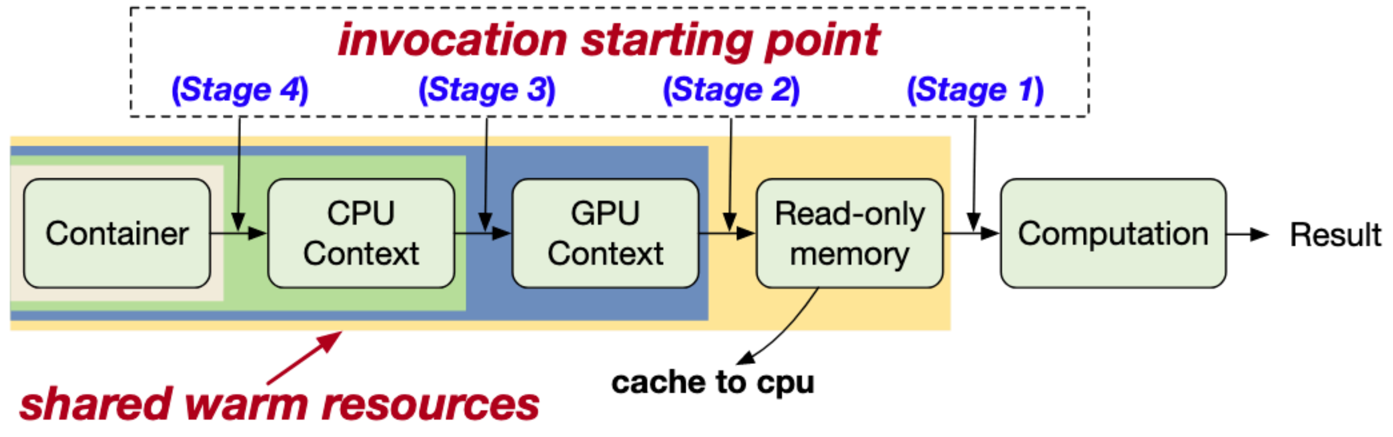


- Solutions:
  - C1: Preserving the serverless programming model -> support runtime
    - shim is inserted to interpose and intercept every API call which, through RPC, are executed at a remote server (API server)
  - C2: Preserving the performance promise of GPU acceleration:
    - Startup optimization: Maintain pool of initialized API servers with initialized GPU runtime
    - Pooling of descriptors on the guest side for functions that don’t need api execution -> avoid unnecessary calls
    - Runtime directly emulates some GPU APIs without remoting them
  - C3: Balancing load and maximizing utilization of remote GPUs.
    - Moving the execution of an application from one GPU to another by monitoring API server assignments and utilization

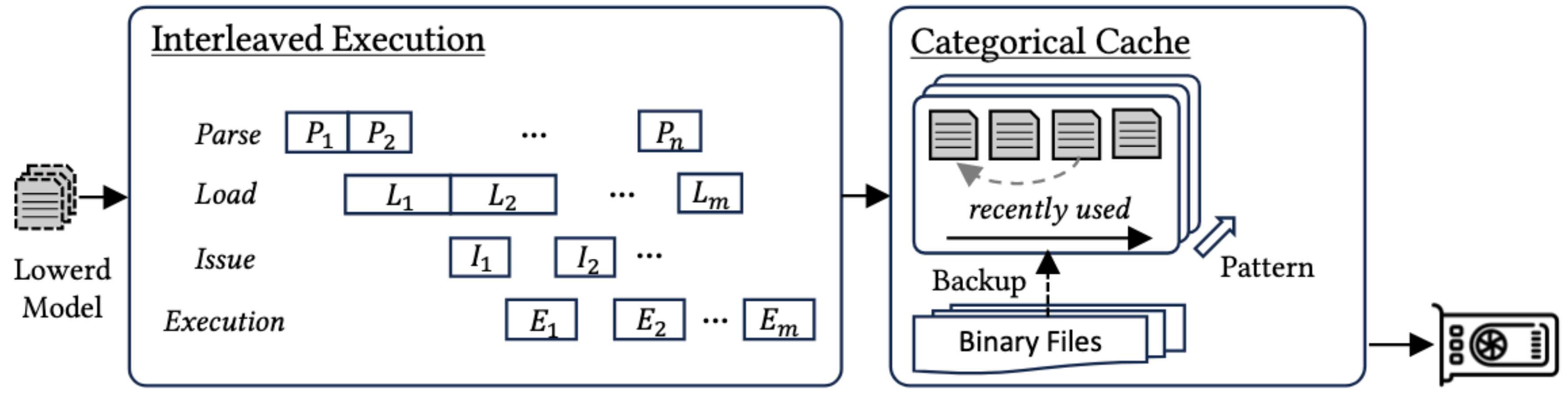
SAGE

- Idea: Data that is loaded through PCIe into GPU can be known before execution (it is usually sent with the request)
  - parallelizing GPU context creation & data preparation
- Sharing-based memory management
  - sharing read-only memory (e.g. ML model weights)
- Keeping context „warm“: multi stage exit
  - How to restore the data/context?





PaSK



- Idea:
  - Offline preparation: DNN-model is submitted once & then optimized.
    - -> DNN layers are converted to sequence of CUDA kernels
  - Online inference serving: Inference request arrives & necessary kernels have to be executed
    - Problem: Kernel needs to be loaded on GPU memory, this is usually done lazily -> waiting for code to be present takes long
    - -> load models immediately after parsing relevant section + categorial cache for layers