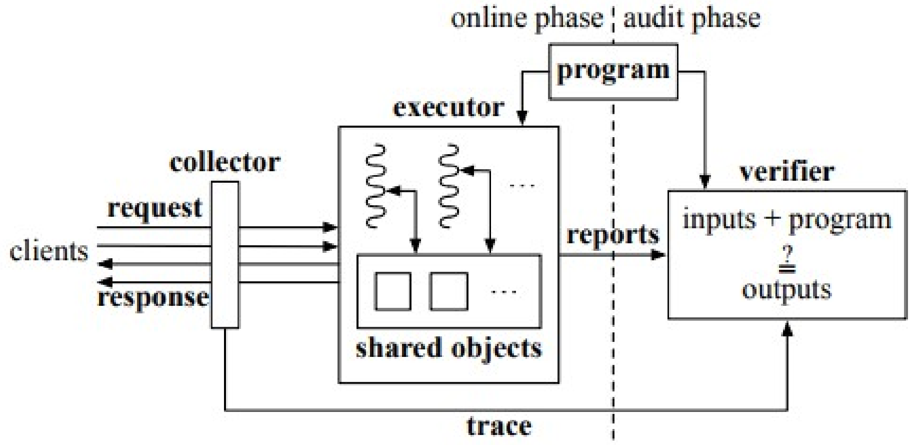
The Efficient Server Audit Problem, Deduplicated Re-execution, and the Web

Cheng Tan, Lingfan Yu, Joshua B. Leners, and Michael Walfish

The Efficient Server Audit Problem

- Clients issue requests(inputs) to the executor and receive responses(outputs)
- Collector captures trace(accuratel
- Executor maintains reports(untrus
- Verifier is responsible for audit pro
- Verifier is weaker than the execut
- Executor is permitted to handle multiple requests at the same time
- Shared objects: DB



The Efficient Server Audit Problem

- Design the verifier and the reports to meet these properties
    - Completeness
        - Verifier must accept the given trace if the executor executed the given program
    - Soundness
        - Verifier must reject if the executor misbehaved during the time period of the trace
    - Efficiency
        - Verifier must require only a small fraction of the computational resources

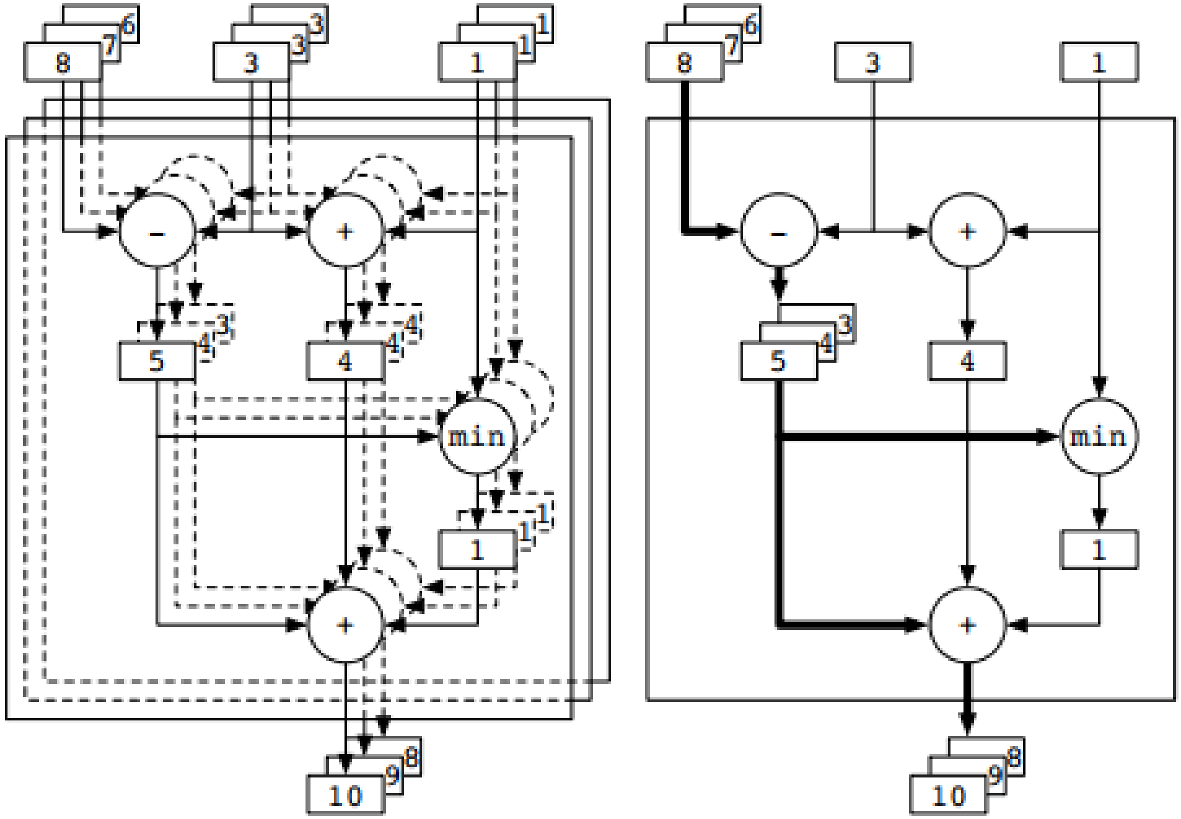
A Solution:SSCO

- Control flow groupings
  - An opaque tag that purportedly identifies the control flow of the execution
  - Requests that induce the same control flow are supposed to receive the same tag
- Operation logs
  - For each shared object, the executor maintains an ordered log of all operations(across all requests)
- Operation counts
  - For each request execution, the executor records the total number of object operations that is issued.

SIMD-on-demand execution

- For each control flow group, verifier conducts a single “superposed” execution that logically executes all requests in that group together
- Instructions whose operands are different across the separate logical executions are performed separately
- Instruction executes only once if operands are same

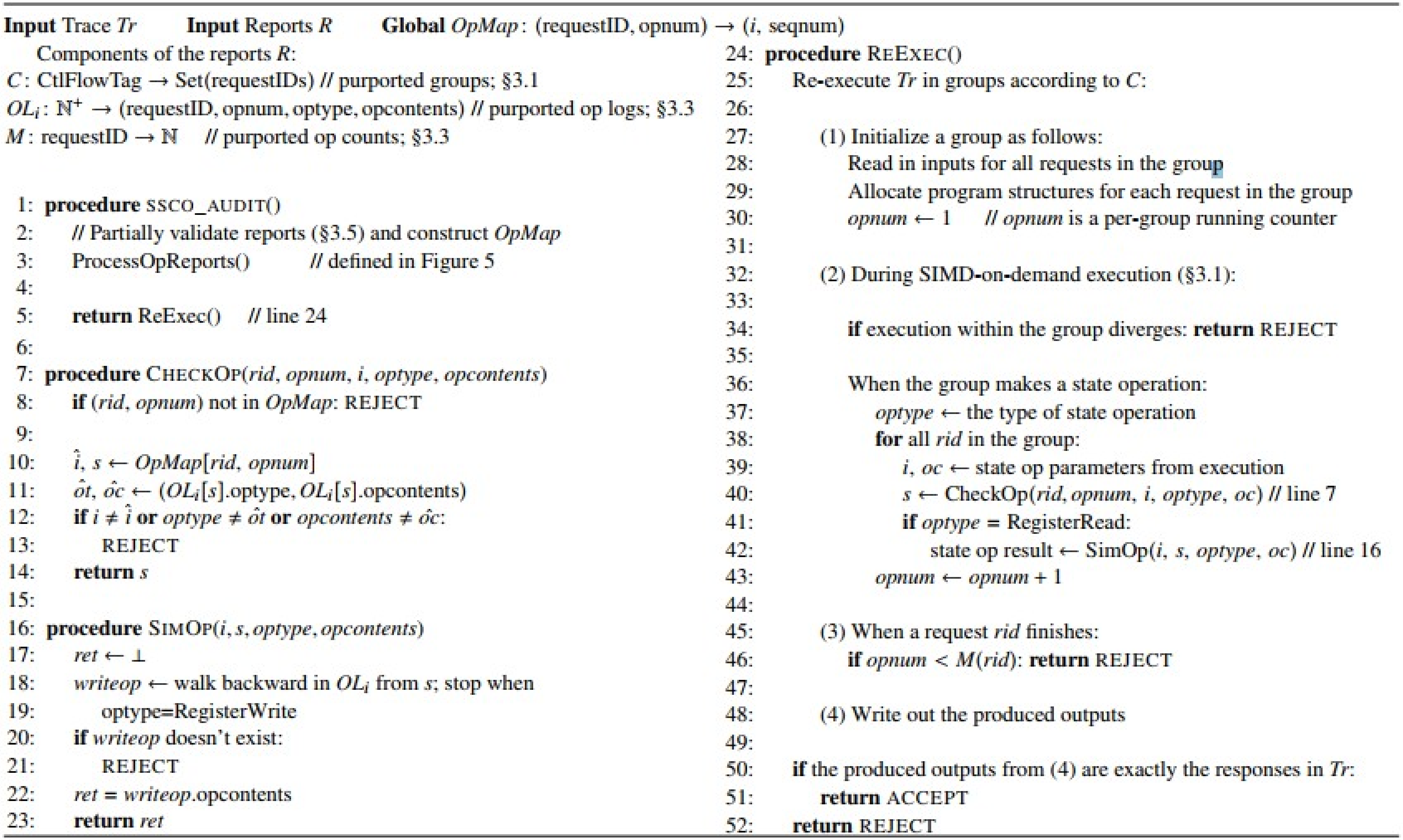
SIMD-on-demand execution



repeated executions

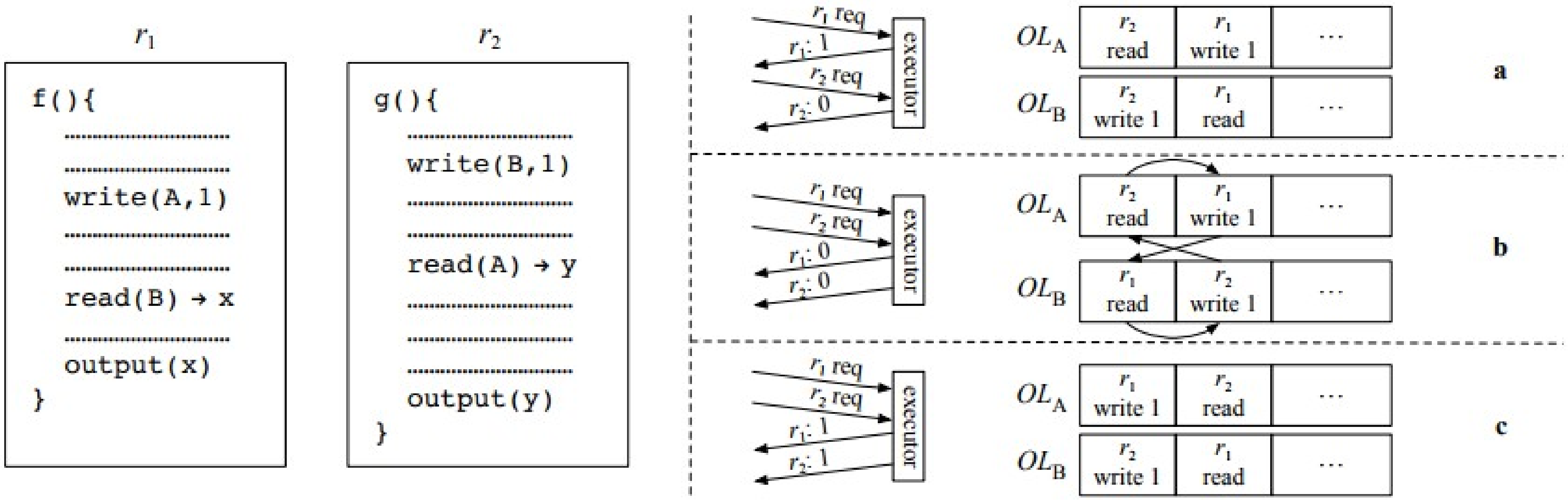
SIMD-on-demand

Simulate and check



```
Input Trace Tr     Input Reports R     Global OpMap: (requestID, opnum) → (i, seqnum)
   Components of the reports R:
C: CtlFlowTag → Set(requestIDs) // purported groups; §3.1
$OL_i$: $\mathbb{N}^+$ → (requestID, opnum, optype, opcontents) // purported op logs; §3.3
M: requestID → $\mathbb{N}$     // purported op counts; §3.3

 1: procedure SSCO_AUDIT()
 2:     // Partially validate reports (§3.5) and construct OpMap
 3:     ProcessOpReports()          // defined in Figure 5
 4:
 5:     return ReExec()     // line 24
 6:
 7: procedure CheckOp(rid, opnum, i, optype, opcontents)
 8:     if (rid, opnum) not in OpMap: REJECT
 9:
10:     $\hat{i}$, s ← OpMap[rid, opnum]
11:     $\hat{ot}$, $\hat{oc}$ ← ($OL_{\hat{i}}$[s].optype, $OL_{\hat{i}}$[s].opcontents)
12:     if $i \neq \hat{i}$ or $optype \neq \hat{ot}$ or $opcontents \neq \hat{oc}$:
13:         REJECT
14:     return s
15:
16: procedure SimOp(i, s, optype, opcontents)
17:     ret ← ⊥
18:     writeop ← walk backward in $OL_i$ from s; stop when
19:         optype=RegisterWrite
20:     if writeop doesn't exist:
21:         REJECT
22:     ret = writeop.opcontents
23:     return ret
24: procedure ReExec()
25:     Re-execute Tr in groups according to C:
26:
27:         (1) Initialize a group as follows:
28:             Read in inputs for all requests in the group
29:             Allocate program structures for each request in the group
30:             opnum ← 1     // opnum is a per-group running counter
31:
32:         (2) During SIMD-on-demand execution (§3.1):
33:
34:             if execution within the group diverges: return REJECT
35:
36:             When the group makes a state operation:
37:                 optype ← the type of state operation
38:                 for all rid in the group:
39:                     i, oc ← state op parameters from execution
40:                     s ← CheckOp(rid, opnum, i, optype, oc) // line 7
41:                     if optype = RegisterRead:
42:                         state op result ← SimOp(i, s, optype, oc) // line 16
43:                 opnum ← opnum + 1
44:
45:         (3) When a request rid finishes:
46:             if opnum < M(rid): return REJECT
47:
48:         (4) Write out the produced outputs
49:
50:     if the produced outputs from (4) are exactly the responses in Tr:
51:         return ACCEPT
52:     return REJECT
```

Simulate and check is not enough



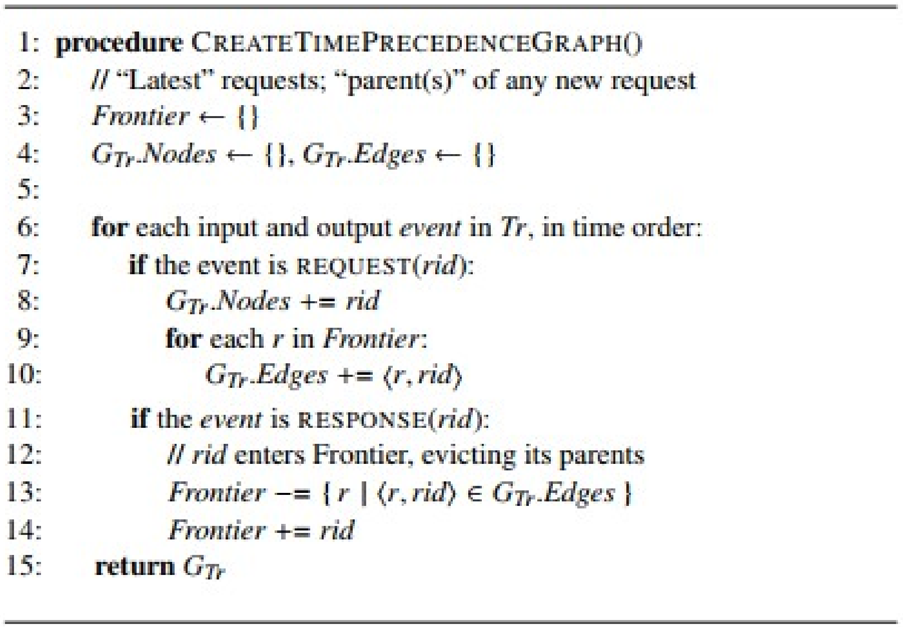
r_1

```
f(){
   ..............................
   ..............................
   write(A,1)
   ..............................
   ..............................
   read(B) → x
   ..............................
   output(x)
}
```

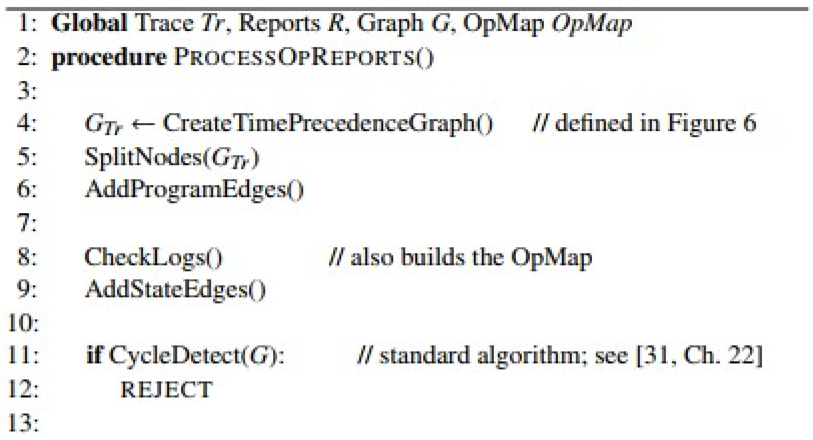
r_2

```
g(){
   ..............................
   write(B,1)
   ..............................
   ..............................
   read(A) → y
   ..............................
   ..............................
   ..............................
   output(y)
}
```

Consistent ordering verification



```
procedure CreateTimePrecedenceGraph()
    // “Latest” requests; “parent(s)” of any new request
    $Frontier \leftarrow \{\}$
    $G_{Tr}.Nodes \leftarrow \{\}$, $G_{Tr}.Edges \leftarrow \{\}$

    for each input and output event in Tr, in time order:
        if the event is REQUEST(rid):
            $G_{Tr}.Nodes$ += rid
            for each r in Frontier:
                $G_{Tr}.Edges$ += $\langle r, rid \rangle$
        if the event is RESPONSE(rid):
            // rid enters Frontier, evicting its parents
            Frontier −= { r | $\langle r, rid \rangle \in G_{Tr}.Edges$ }
            Frontier += rid
    return $G_{Tr}$
```

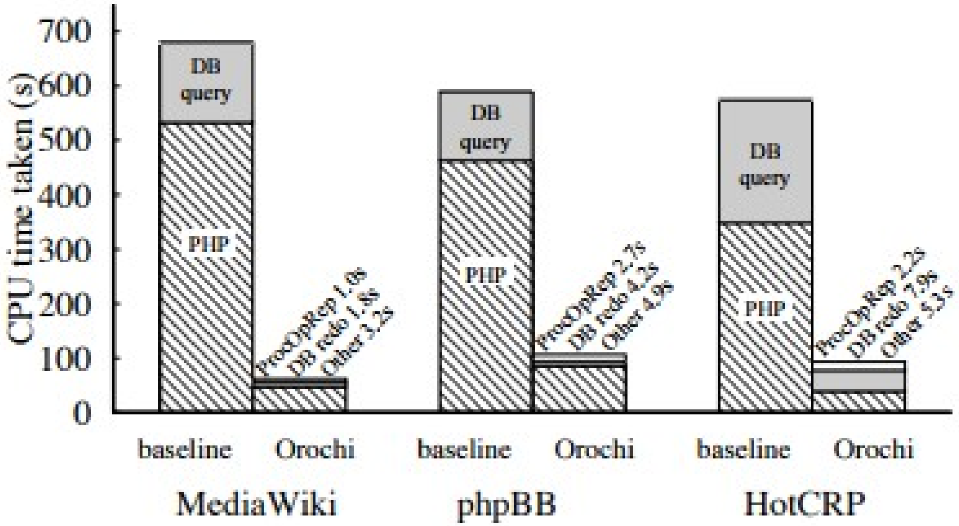


```
Global Trace Tr, Reports R, Graph G, OpMap OpMap
procedure ProcessOpReports()

    $G_{Tr} \leftarrow$ CreateTimePrecedenceGraph()      // defined in Figure 6
    SplitNodes($G_{Tr}$)
    AddProgramEdges()

    CheckLogs()              // also builds the OpMap
    AddStateEdges()

    if CycleDetect(G):         // standard algorithm; see [31, Ch. 22]
        REJECT

```

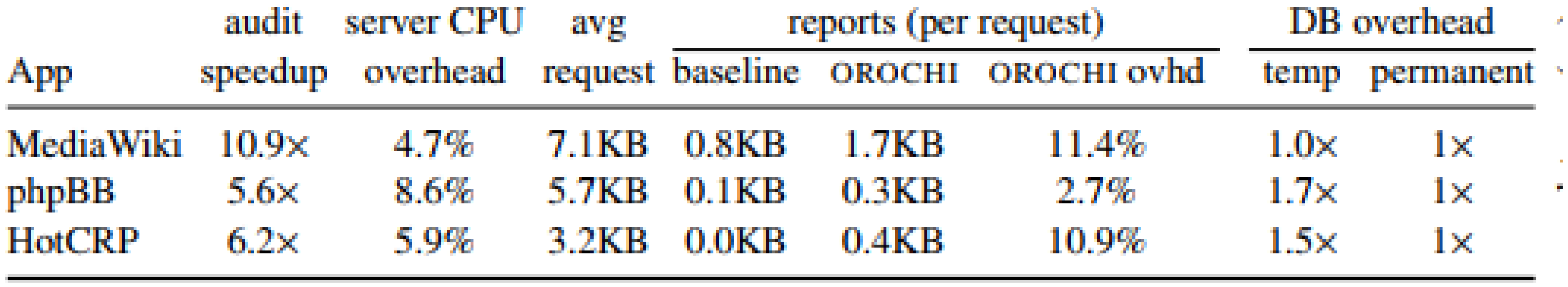
OROCHI

- Orochi targets apps based on PHP and SQL(LAMP)
- Server and verifier: modified PHP runtimes
- Built atop HipHop VM
- 20K lines of C++, PHP, Bash, Python
- Applications
    - MediaWiki, phpBB and HotCRP

Orochi’s verifier is efficient



The price of verifiability is tolerable



	audit	server CPU	avg	reports (per request)			DB overhead	
App	speedup	overhead	request	baseline	OROCHI	OROCHI ovhd	temp	permanent
MediaWiki	10.9×	4.7%	7.1KB	0.8KB	1.7KB	11.4%	1.0×	1×
phpBB	5.6×	8.6%	5.7KB	0.1KB	0.3KB	2.7%	1.7×	1×
HotCRP	6.2×	5.9%	3.2KB	0.0KB	0.4KB	10.9%	1.5×	1×