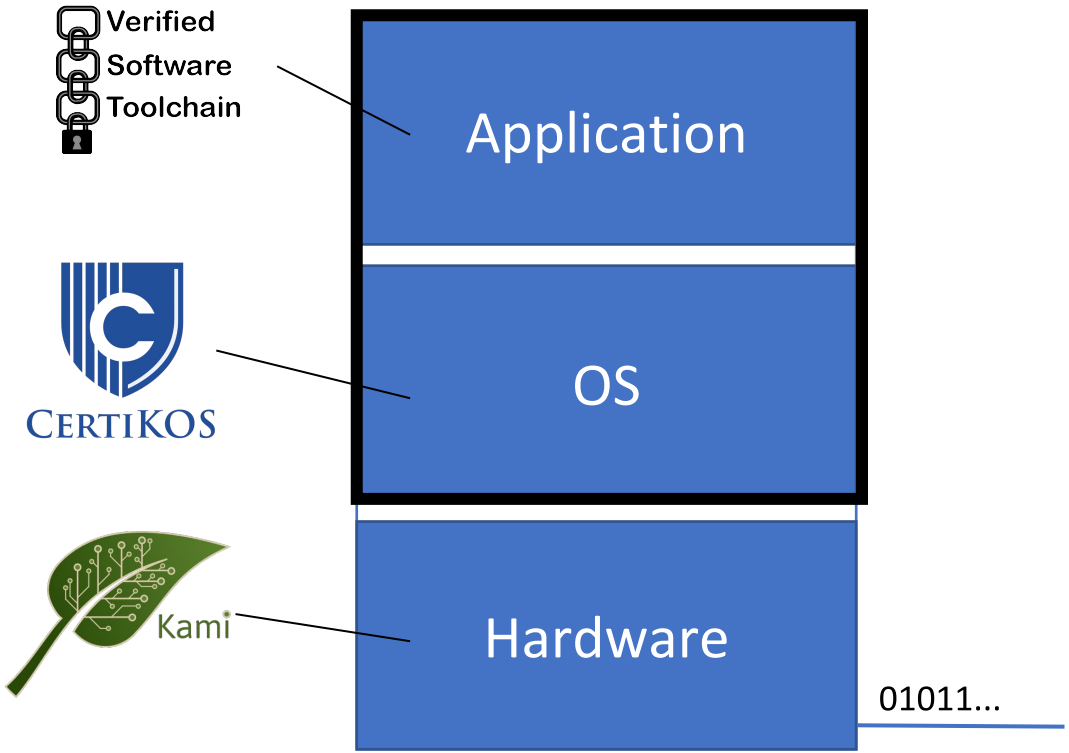
Bridging the Specification Gap Between VST and CertiKOS

William Mansky (UIC), Wolf Honore (Yale)

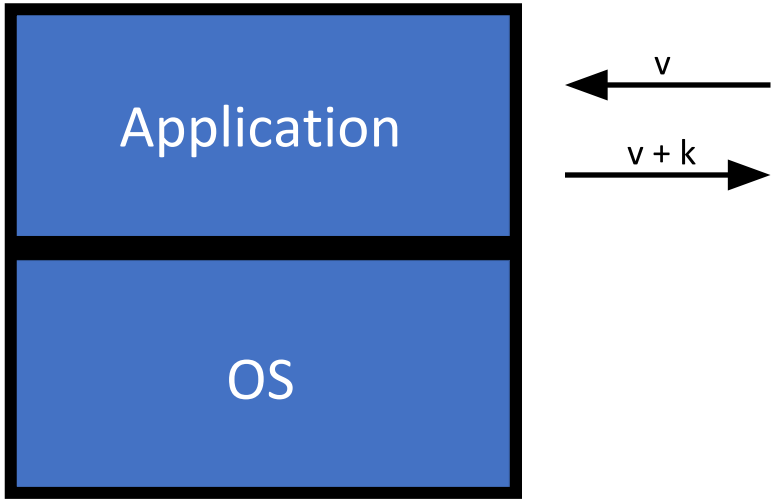
At DeepSpec 2019

Verification from RFCs to transistors



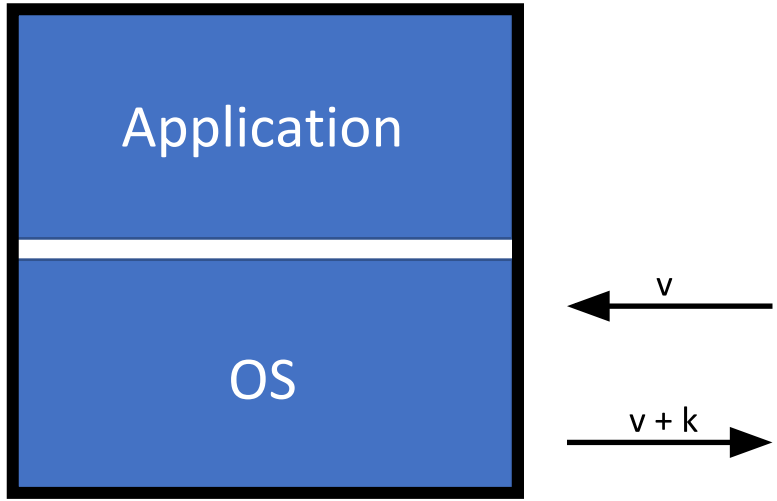
Communicating Programs and the OS

```
x = recv();
…
send(x + k);
```



Communicating Programs and the OS

```
x = recv();
…
send(x + k);
```

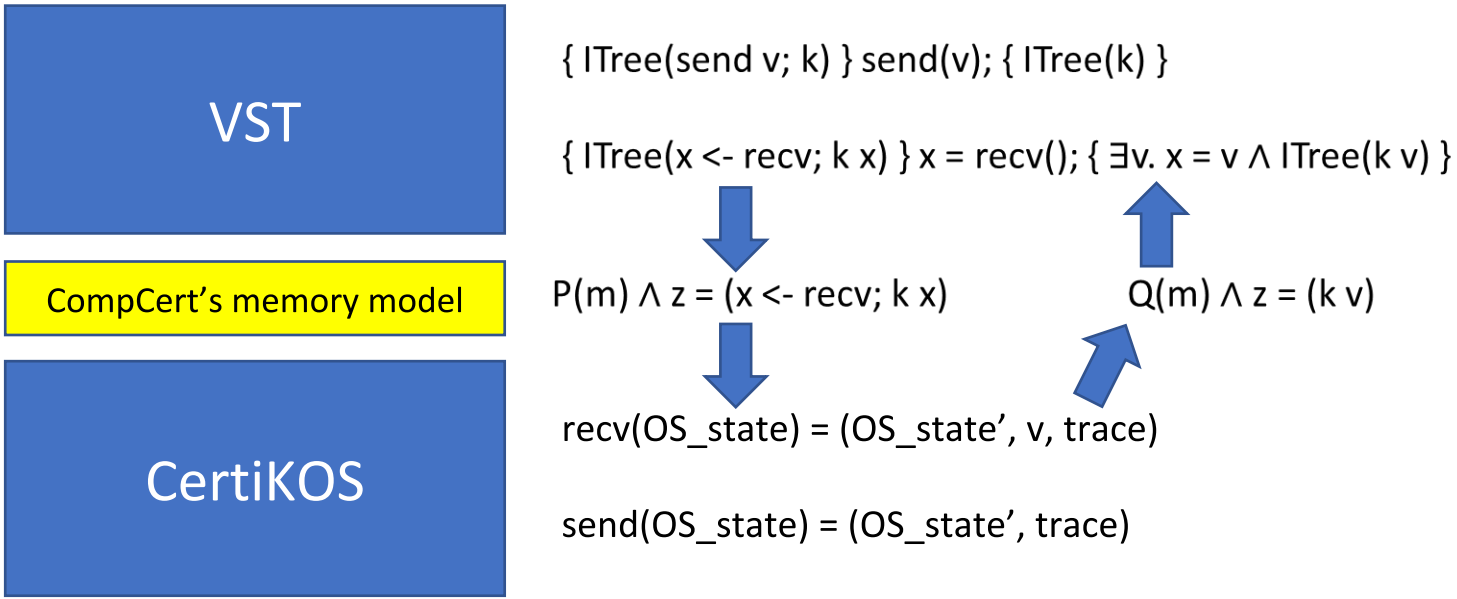



```
recv:
   …
send:
   …
```

Bridging the Specification Gap

- Extended VST with external state and proved a new soundness theorem
- Developed a technique for proving that CertiKOS system calls satisfy VST specs
- Connected some common system calls (socket send/recv, putchar/getchar) and used them to verify simple communicating programs

Connecting VST and CertiKOS Specifications



Lowering VST Specifications

$[\![$ data $\mapsto v_1,\ldots, v_N$ * ITree(write $(v_1 + \ldots + v_N))]\!]$(r, ext) =

$[\![$ data $\mapsto v_1,\ldots, v_N]\!]$(r) $\wedge$ ext = write $(v_1 + \ldots + v_N)$

Lowering VST Specifications

$[\![$ data $\mapsto v_1,\ldots, v_N$ * ITree(write $(v_1 + \ldots + v_N)$) $]\!]$(r, ext) =

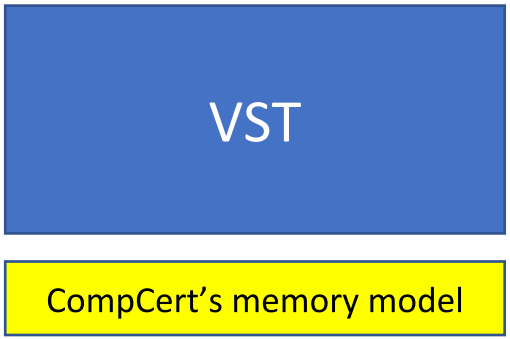
$[\![$ data $\mapsto v_1,\ldots, v_N$ $]\!]$(r) $\wedge$ write $(v_1 + \ldots + v_N) \sqsubseteq$ ext

m(data) = $v_1 \wedge \ldots \wedge$ m(data + N – 1) = $v_N \wedge$ write $(v_1 + \ldots + v_N) \sqsubseteq$ ext

{ ITree(write v; k) } write(v); { ITree(k) }

{ buf $\mapsto$ msg * ITree(send msg; k) } send(buf); {buf $\mapsto$ _ * ITree(k) }

Lowering VST Specifications



{ ITree(send v; k) } send(v); { ITree(k) }

{ ITree(x <- recv; k x) } x = recv(); { ∃v. x = v ∧ ITree(k v) }

P(m) ∧ z = (x <- recv; k x)

Q(m) ∧ z = (k v)

```
int data[N];
int c;
{ data ↦ _ * ITree(v1 <- read; ...; vN <- read; write (v1 + ... + vN)) }
for(int i = 0; i < N; i++){
   { data ↦ v1,..., vi−1 * ITree(vi <- read; ...; vN <- read; write (v1 + ... + vN)) }
   c = read();
   { data ↦ v1,..., vi−1 * ITree(vi+1 <- read; ...; vN <- read; write (v1 + ... + vN)) }
   data[i] = c;
   { data ↦ v1,..., vi * ITree(vi+1 <- read; ...; vN <- read; write (v1 + ... + vN)) }
}
{ data ↦ v1,..., vN * ITree(write (v1 + ... + vN)) }
write(sum(data, N));
{ data ↦ v1,..., vN * ITree() }
```

VST Soundness with External State

- For each external call with spec {P} f(); {Q}, write a CompCert-level spec P', Q' such that
  - P(r, z) ⇒ P'(dry(r), z)
  - Q'(m, z) ⇒ Q(reconstruct(r, m), z)

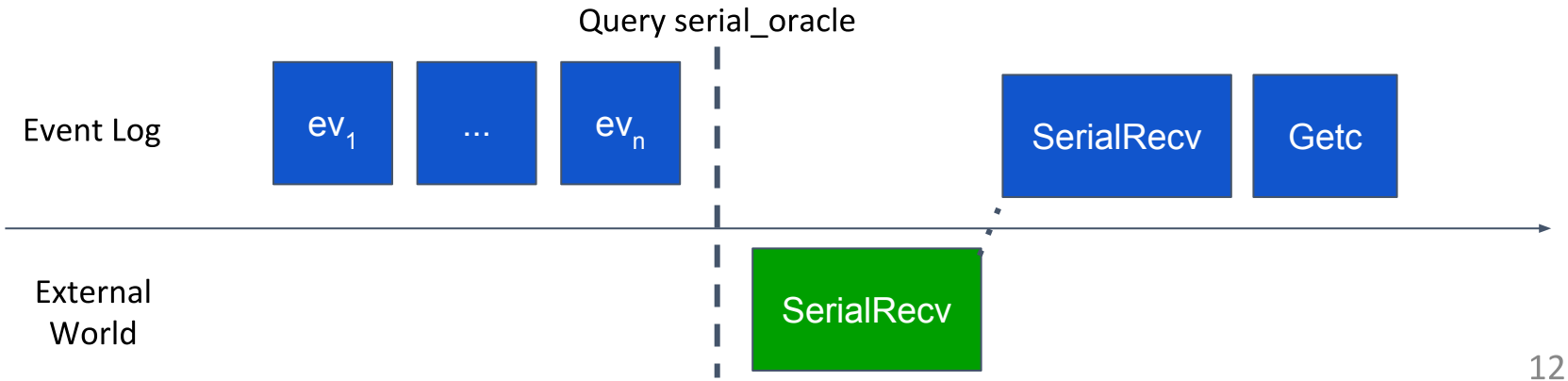
- Soundness theorem: if a program P using external functions $f_1, \ldots, f_n$ is verified with external specs $J_1, \ldots, J_n$, and each J_i corresponds to a dry spec D_i, then P executes correctly with any implementation of $f_1, \ldots, f_n$ that satisfy $D_1, \ldots, D_n$

CertiKOS Specifications

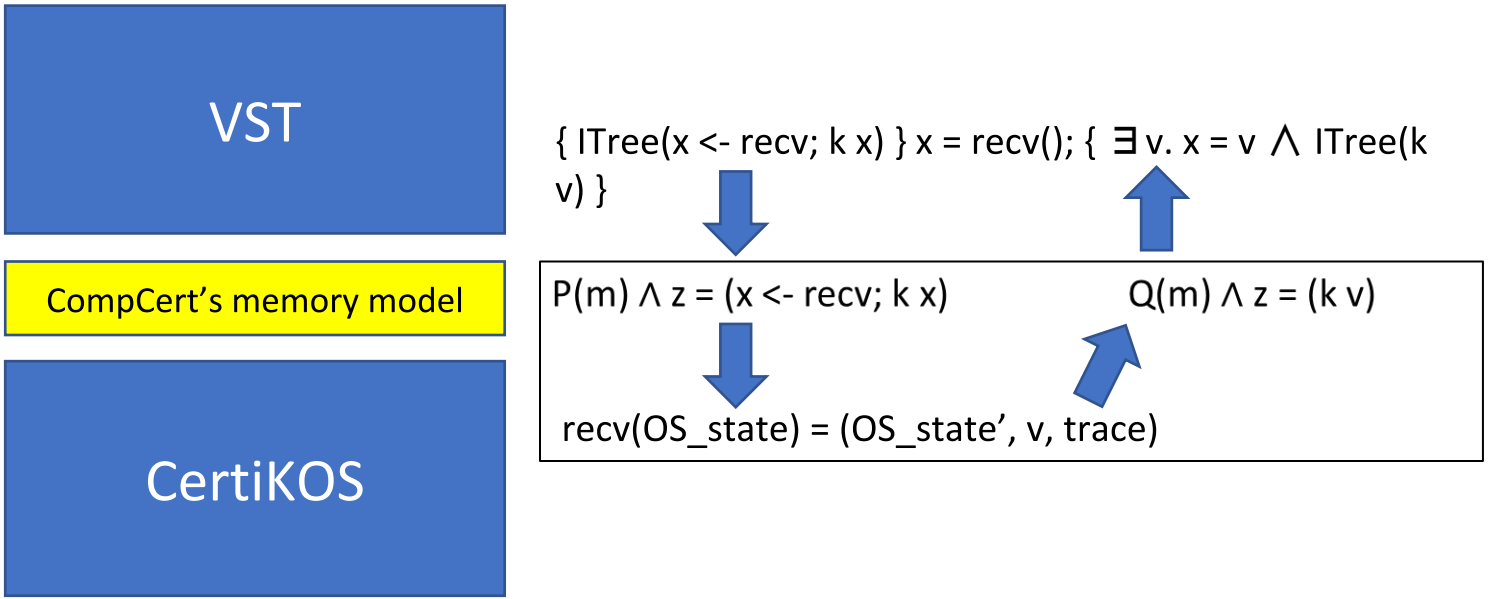
```
Parameter serial_oracle : list event -> event.

Definition serial_in_spec (st : OSState) : OSState * Z :=
  ... (* read buffers, compare bits, etc *)
  let new := serial_oracle st.(serial_log) in
  match new with
  | SerialRecv data =>
    let (st', c) := ... in (* process data *)
    (st'/[serial_log := st.(serial_log) ++ [new]], c)
  | _ => ... (* handle other events *)
  end.
```

CertiKOS Specifications

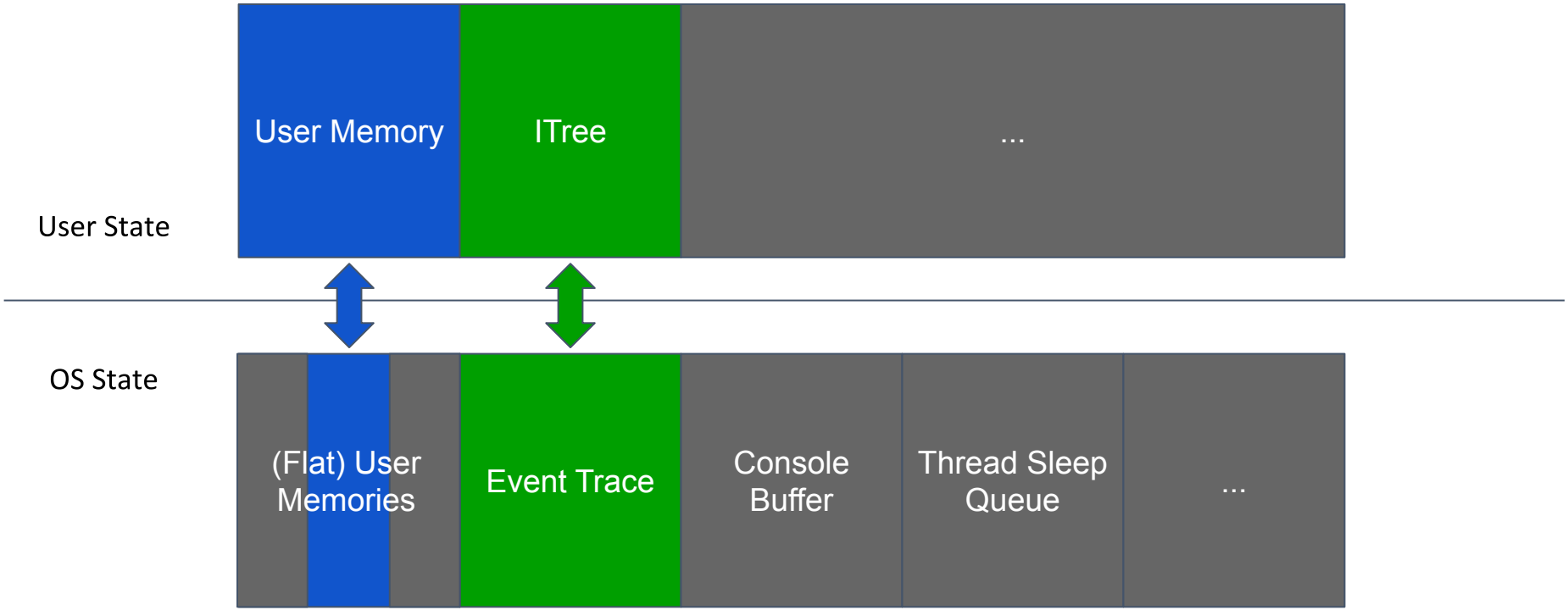
```
Definition sys_getc_spec (st : OSState) : OSState * Z :=
  let st' := ... (* check for interrupts *)
  match st'.(console_buf) with
  | c :: rest =>
    (st'/[console_buf := rest]/[serial_log := st.serial_log ++ [Getc c]], c)
  | nil => ... (* return error code *)
  end.
```



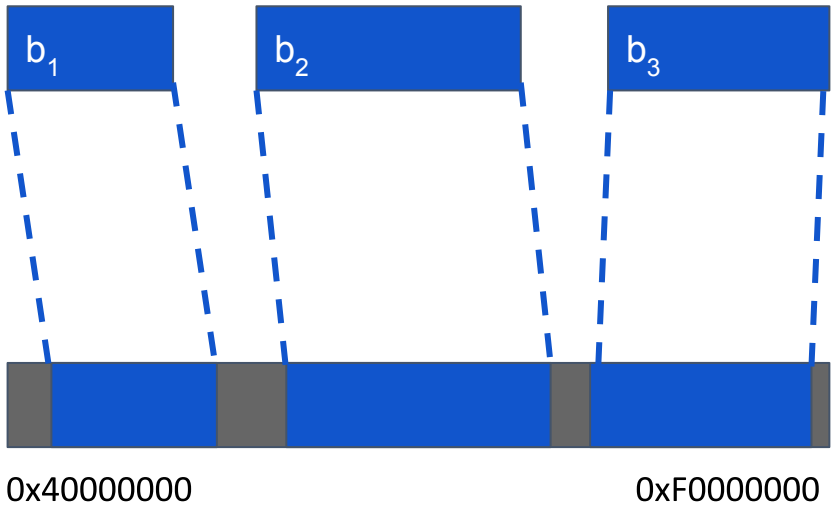
Connecting Dry Specifications to CertiKOS



Relating VST and CertiKOS States



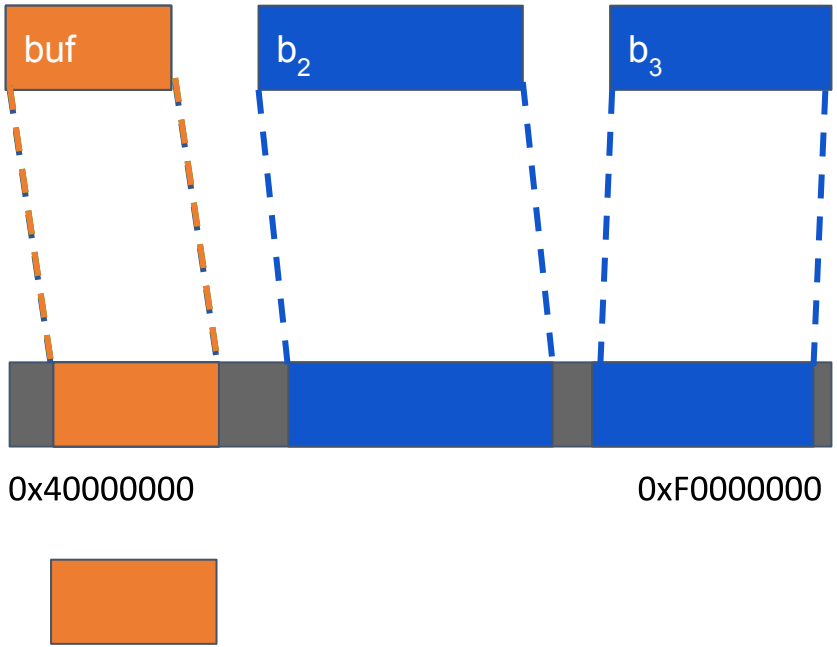
Relating Memories



Relating Memories

```
{buf ↦ _ * ITree(cs <- readN len; k cs)}
int get_bytes(int len, char *buf);
{buf ↦ cs * ITree(k cs)}
```

- Assume precondition on buf block
- Map buf block to its flat virtual address
- OS reads bytes into internal buffer
- OS copies bytes into flat user memory
- Translate virtual address back into block
- Prove postcondition on buf block



Relating External Events

VST specs consume ITrees

```
{ITree(c <- read; k c)}
int getc();
{ITree(k c)}
```

CertiKOS specs produce traces

```
Definition sys_getc_spec (st : OSState) : OSState * Z :=
  let st' := ... (* check for interrupts *)
  match st'.(console_buf) with
  | c :: rest =>
    (st'/[console_buf := rest]/[serial_log := st.serial_log ++ [Getc c]], c)
  | nil => ... (* return error code *)
  end.
```

Relating External Events

ITrees to sets of traces

```
ITree(c <- read; k c) → { Read c ++ tr | ∀tr ∈ traces_of (k c)
}
```

Extract newly generated user-visible events

```
let tr := st.(serial_log) in
let tr' := (sys_getc_spec st).(serial_log) in
filter user_visible (strip_common_prefix tr tr')
```

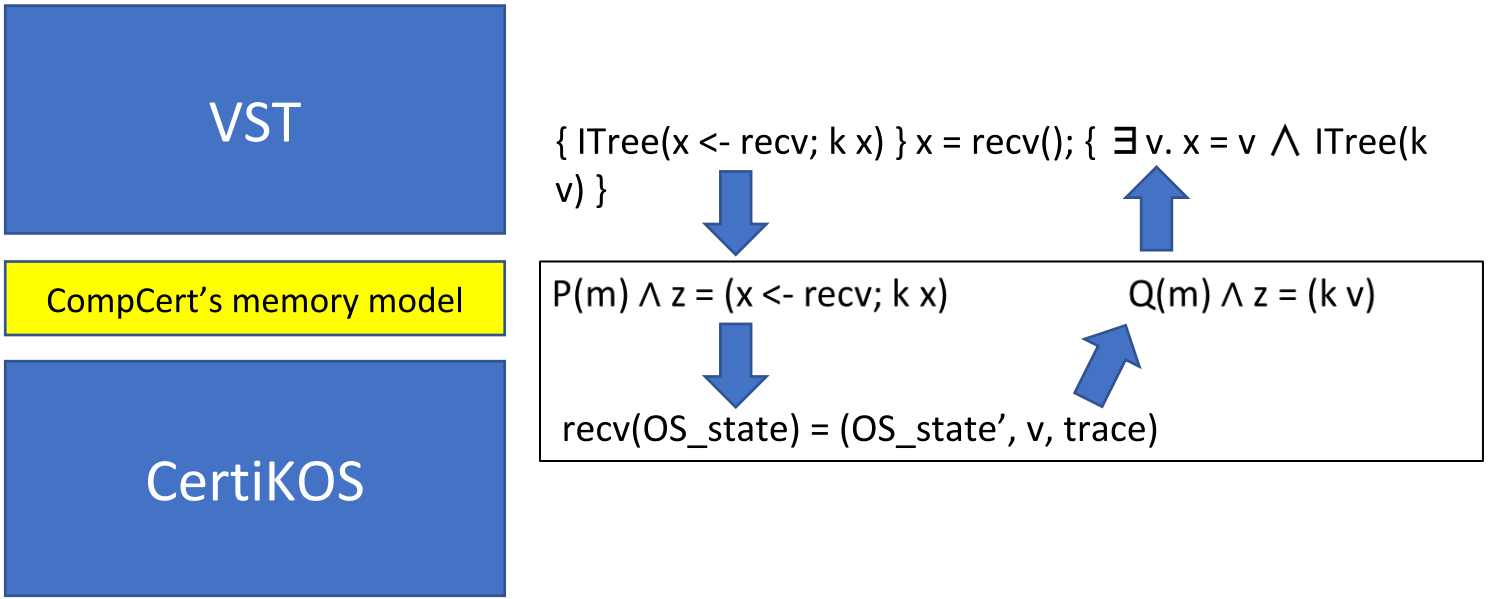
Relating External Events

Relate pre- and postcondition ITrees to CertiKOS-generated trace

```
∀tree tree' tr.
  ∀tr' ∈ traces_of tree' ⇒
  (tr ++ tr') ∈ traces_of tree
```


```
{ITree(c <- read; k c)}

int getc();

{ITree(k c)}
```

Connecting Dry Specifications to CertiKOS



Top-Level Theorem

- VST soundness: verified programs execute correctly in CompCert C semantics, using dry specs for external calls
- CertiKOS correctness: system call specs implement dry specs
- Combined: verified programs execute correctly in CertiKOS
  - But this isn't yet proved in Coq, and might be more gaps to bridge between VST and CertiKOS main theorems

Conclusion

- Our verified C programs no longer need to assume the correctness of I/O system calls
- Works for console I/O, network sockets, files, …
- First step of connection between VST and CertiKOS

Future work:

- All the system calls for web server, file system, …
- Top-level theorem for VST-verified programs running on CertiKOS