2nd Iris Workshop, May 2022,
Radboud University Nijmegen, The Netherlands

LOCAL STATES

More hidden states, more modalities, and generalized invariants and ghost ownership

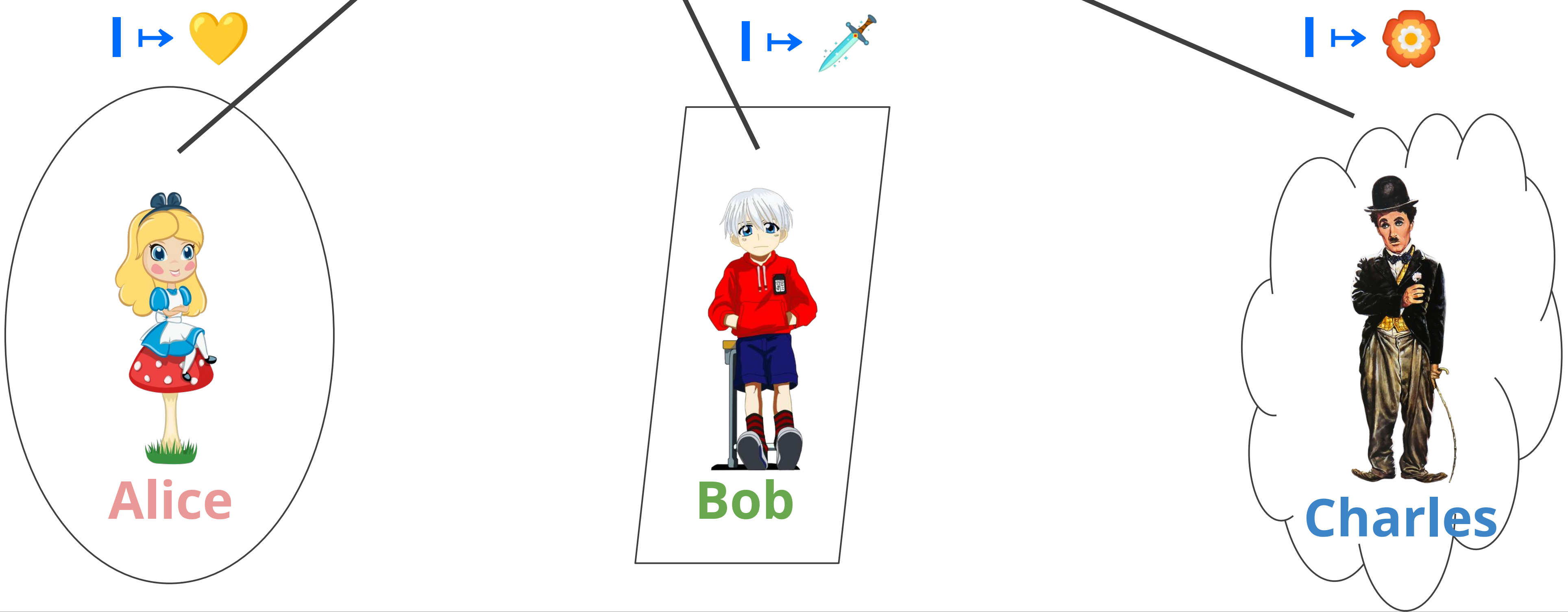
Hai Dang, Gregory Malecha, David Swasey

Local states?

Usually, l ↦ v means t

differently depending on one's local states!

But everyone has *secrets*!



Interpretation depends on local states

- not uncommon
  - in weak memory
  - in distributed systems
  - in virtual address spaces

$l \mapsto v$ depends on

- local caches
- local node states
- local page tables

- a solution: indexing resources by the local state $l \mapsto\{s\}\ v$

Alice

$l \mapsto\{A\}\ v :\approx$

$l \mapsto$ 💛

Bob

$l \mapsto\{B\}\ v :\approx l$

$\mapsto$ 🗡️

Charles

$l \mapsto\{C\}\ v :\approx$

$l \mapsto$ 🏵️

Indexing resources by local state: P s

- but not everything depends on some local states—local states should be *ambient*
  - hide local states with Iris' **monPred**
  - only work with local states explicitly when needed
    - using ***modalities***, inspired by Iris-based weak-mem works
- benefits:
  - cleaner for things that don't care about local states
  - more idiomatic reasoning with explicit local states

examples coming in a moment …

But what about *composing* local states?

weak-mem ➕ virtual address ➕ ???

- how to compose a logic that knows about weak-mem and a logic that knows about virtual address?
- open world problem, similar to `inG`

WIP: Monotone Lenses

weak-mem ✚ virtual address ✚ ???

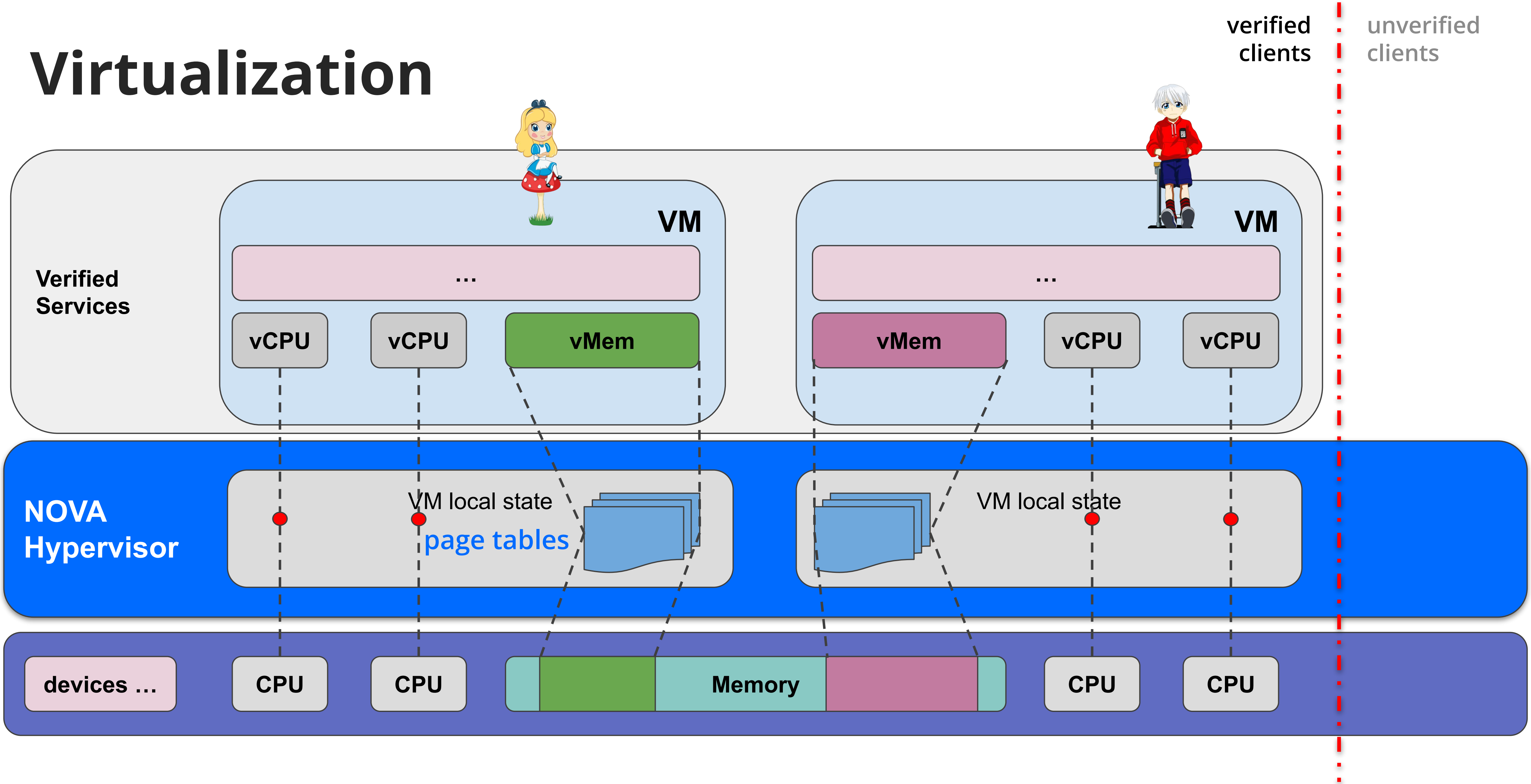
solution:

1. generalizing monPred with *monotone lenses* to encode that the local state type embeds some concrete local state type
2. generalizing monPred modalities to lens-induced *families* of modalities
3. generalizing invariants and ghost ownership

BedRock Systems Inc

EXAMPLE:
VIRTUAL ADDRESS SPACES

Virtualization



BedRock Systems Inc

Example: virtual address spaces

➡ goal: building a points-to ownership for a VM's memory

$va \mapsto v \quad :\approx \; \exists\; pa,\; va \mapsto_{AS} pa * pa \mapsto v$

- va : virtual machine address
- pa : physical machine address

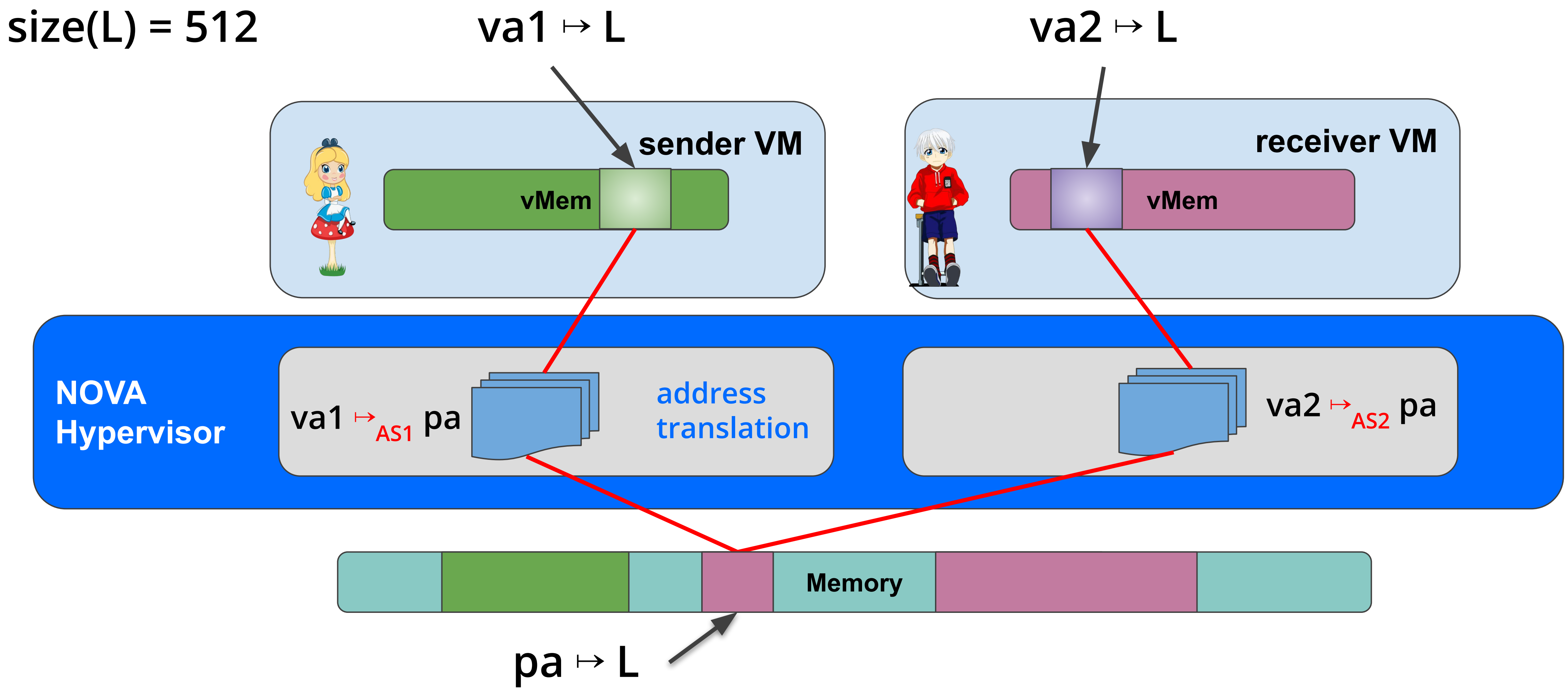
$va \mapsto_{AS} pa$:≈ in the *current* virtual address space AS,
va is mapped to pa

Page tables are local states of an address space

Building virtual points-to

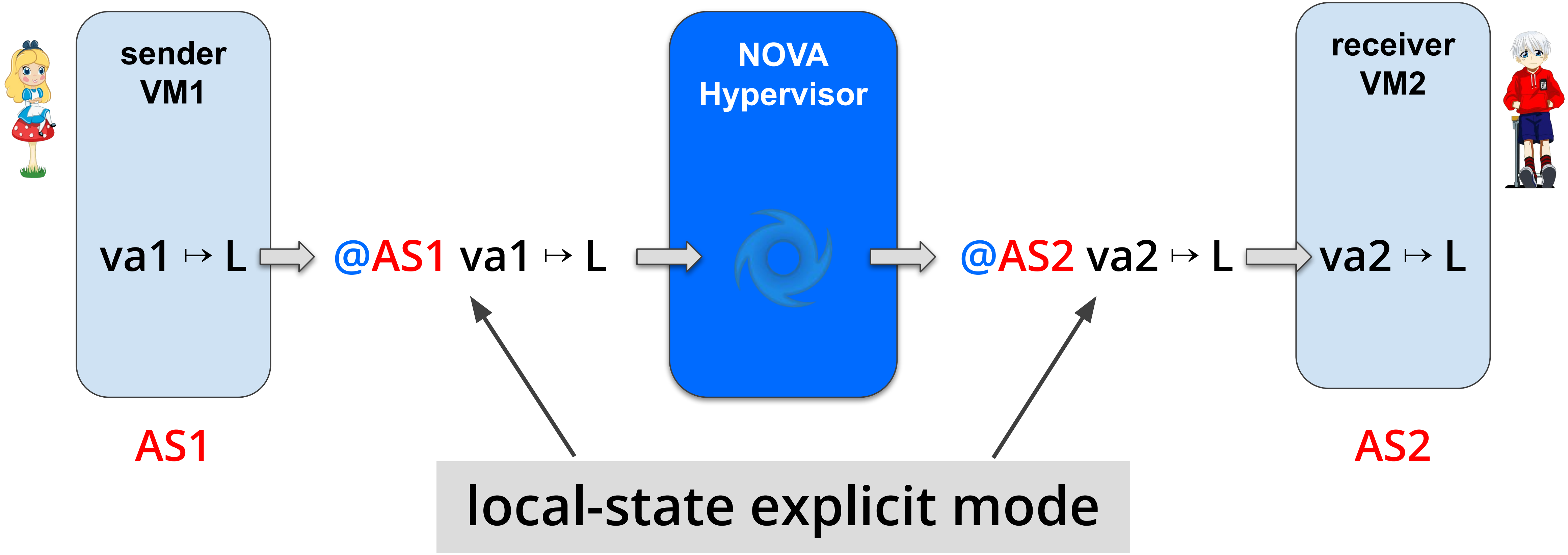
- aProp := monPred AddrSpace iProp
  - ≈ AddrSpace → iProp, but "monotone"
- $va \mapsto v$: aProp :≈ λ AS, $\exists$ pa, $va \mapsto_{AS} pa \ast pa \mapsto v$
- for those *without* interesting interaction with AS, same rules:
  - $\{ va \mapsto v \} \ast va \{ w.\ w = v \ast va \mapsto v \}$
- *with* interesting interactions with page tables, use ⎕AS and @AS P
  - state-explicit *modalities* inspired by Iris-based weak-mem works

Example: send memory across VMs



Logically,

size(L) = 512



Quick summary: a recipe for local states in Iris

- tProp := monPred I iProp, where I is the type of the local state
- for those *without* interesting interactions with the local state,
  - lift rules for tProp
  - pro: same rules as before
- for those *with* interesting (non-local) interactions,
  - explicit reasoning with local-state modalities (eg., ∃i, @i P)
  - *spoiler*: adjustments for invariants and ghost ownership

MONOTONE LENSES

Composing local states

Composing local states?

weak-mem + virtual address + ???

weak-mem	virtual address	???
vProp	aProp	
logic that knows about weak-mem states	logic that knows about address translation	future logics

➔ **idea:** work with some general tProp and, as needed, assume that tProp "embeds" vProp and/or aProp and/or others.

Generalizing the local state

➔ **idea:** work with some general tProp and, as needed, assume that tProp "embeds" vProp and/or aProp and/or others.

- idea in "typeclass" style:
  - have tProp as the assertion type
  - need weak-mem? assume HasVProp tProp
  - need address space? assume HasAProp tProp
- *lightweight* implementation with monPred
  - quantify over arbitrary local state type I: ∀ I, monPred I iProp (≈ tProp)
  - when needed, assume I "embeds" View (weak-mem states) and/or AddrSpace (address-translation states)

Monotone Lenses

```
Context {I : biIndex} {PROP: bi}.
Notation tProp := monPred I PROP.
```

“I embeds the address-translation states AddrSpace“

≈ the existence of a *monotone lens* from I into AddrSpace

Context {L_{AS} : I -ml> AddrSpace}.

💡 Monotonicity to fit monPred, crucial for stability of the frame.

Monotone Lenses

Context {I : biIndex} {PROP: bi}.

Notation tProp := monPred I PROP.

Context {J : biIndex} {L : I -ml> J}.

```
Structure MLens (I J : biIndex) : Type := MLensMake {
 mlens_get : I -> J ;
 mlens_set : J -> I -> I ;
 mlens_get_set : ∀ i j, mlens_get (mlens_set j i) = j ;
 mlens_set_get : ∀ i, mlens_set (mlens_get i) i = i ;
 mlens_set_set : ∀ i j1 j2, mlens_set j1 (mlens_set j2 i) = mlens_set j1 i ;
 mlens_get_mono : Proper ((⊑) ==> (⊑)) mlens_get ;
 mlens_set_mono : Proper ((⊑) ==> (⊑) ==> (⊑)) mlens_set ;
}.
```

Operations on lenses:

- product/projection
- compose
- equivalence
- disjointness
- inclusion
- L_{id} as a unit

Families of modalities

```
Context {I  : biIndex} {PROP: bi}.

Notation tProp := monPred I PROP.

Context {J : biIndex} {L : I -ml> J}.
```

- Resources that are local-state independent simply ignore I
- Resources that depend on some local-state J use J's family of modalities

@(L,j) P : tProp :≈

P holds at a local state whose J component is j

⊒{L} j : tProp :≈

The current local state's J component is at least j

- + more modalities
- interactions between families

BedRock Systems Inc

INVARIANTS and GHOST OWN

with lenses

Invariants and ghost ownership?

weak-mem + virtual address + ???

vPropr is wrong—

vProp

logic that knows about weak-mem states

aProp

logic that knows about address translation

future logics

tProp

Problem: BI-general invariants and ghost own

- Each lens-dependent monPred benefits from invariants and ghost own
  - Iris invariants and ghost ownership are tied to iProp
- Generalization: BI with own

```
Class HasOwn {PROP : bi} {A : cmra} : Type := {
 own            : gname → A → PROP ;
 own_op         : ∀ γ (a b : A), own γ (a ⋅ b) ⊣⊢ own γ a ∗ own γ b ;
 own_mono       :> ∀ γ, Proper (flip (≼) ==> (⊢)) (own γ) ;
 own_ne         :> ∀ γ, NonExpansive (own γ) ;
 own_timeless :> ∀ γ (a : A), Discrete a → Timeless (own γ a) ;
 own_core_persistent :> ∀ γ a, CoreId a → Persistent (own γ a)
}.
```

```
Class HasOwnValid `{!BiEmbed siPropI PROP} `{!HasOwn PROP A} ...
Class HasOwnUpd `{!BiBUpd PROP} `{!HasOwn PROP A} ...
Class HasOwnUnit `{!BiBUpd PROP} {A : ucmra} `{!HasOwn PROP A} ...
```

➡ more use of siProp

- Generalization: *with* modalities, invariants are general *except* for allocation

```
Definition inv_def N (P : PROP) : PROP :=
   (□ ∀ E : coPset, ⌜↑N ⊆ E⌝ → |={E,E ∖ ↑N}=> ▷ P ∗ (▷ P ={E ∖ ↑N,E}=∗ emp))%I.
```

Invariants for monPred's

- Iris invariants are lifted into objective invariants for monPred
  - resources stored in invariants must be independent of the local state I
  - Objective P := ∀ i1 i2, P i1 ⊢ P i2
  - Objective (@i P)
  - ALLOC: **Objective P** → ▷ P ⊢ |={E}=> inv N P
  - INTRO: P ⊢ ∃ s, ⊒i ∗ @i P
  - ELIM: @i P ⊢ ⊒i -∗ P

- generalization for lenses:
  - ObjectiveWith L P := ∀ (i : I) (j : J), P i ⊢ P i[L := j]
  - P is independent of the J component of the local state
  - *Components whose local states only differ in J can communicate* P *freely*
    - $@(L_{cpu}$, c) l ↦ v can be shared across CPUs that are in the same address space
  - Derived notion of *local invariants*
    - $@(L_{cpu}$, c) l ↦ v can be put in AS-local invariants

Problems

- BI-general invariants and ghost ownership
- Algebra of lenses ❓
  - interactions of families of modalities
- Adequacy of wp ❓
- biIndex that depends on the logic ❓
  - an abstraction from one lens to another that depends on the logic
  - similar to higher-order ghost state
- Cross-BI modalities ❓
  - lenses generalize I in monPred I PROP
  - what about PROP? from monPred I PROP1 to monPred I PROP2?
    - at BedRock, `mpred` to `Rep`.
- Proofmode/Automation ❓

CONCLUSION

- monPred to hide local states and local-state modalities to expose them when needed
- monotone lenses to abstract over and compose local states
- generalized invariants and ghost ownership as useful features of BIs
- work-in-progress, with quite a few TODOs

THANK YOU

Local states?

```
{ isLock(s, P) }
    s.lock();
{ P }
    …
    …
    …
{ isLock(s, P) * P }
    s.unlock();
{ emp }
```

temporarily owning P *locally*

❌ not the kind of "local" we focus on here

Local states?

```
#foo.hpp
static int x = 0;
class foo {
    int f() { … ; x = 1; … }
};
```

compilation-unit local statics

```
#bar.cpp
int bar() {
    int arr[N];
    …
};
```

thread-local stack variables

➡ often modeled as *explicit* resources (points-to)

Implicit local states

Local states are ambient:

1. they are always around/readily available
   - by threading through weakest-pre
   - more abstractly, by using non-atomic invariants

2. they should be unobtrusive
   - by hiding them with Iris' monPred

Example: weak memory in Iris

explicit ownership of the local state

a View—presenting the local cache of a CPU

$$\{\ tid \mapsto V * P\ V\ \}\ e \text{ on } tid\ \{\ (v, V').\ tid \mapsto V' * Q\ V'\ \}$$

- Most things don't care about the view V, only memory accesses do
- In many cases, memory accesses do not have interesting interactions with the view V

➡ Motivation for hiding views, and only let them bubble up when it's "interesting".

Implicit weak memory states in Iris

resources may depend on the local state

$$\{\ tid \mapsto V * P\ V\ \}\ e \text{ on } tid\ \{\ (v, V').\ tid \mapsto V' * Q\ V'\ \}$$

⇓

$$\{\ P\ \}\ e\ \{\ v.\ Q\ \}$$

$P\ Q$: monPred View iProp ≈ View → iProp, but “monotone”

$wp\ e\ \{\ v.\ Q\ \}$: monPred View iProp :≈

$$\lambda V,\ \ \forall\ tid,\ tid \mapsto V \mathrel{-\!\!*} wp\ e \text{ on } tid\ \{\ (v, V').\ tid \mapsto V' * Q\ V'\ \}$$

Implicit weak memory states in Iris

P Q : monPred View iProp ≈ View → iProp, but "monotone"

completely local
unobtrusive

$\{ \text{emp} \}\ z1 + z2\ \{ v.\ \ulcorner v = z1 +_Z z2 \urcorner \}$

$\{ l \mapsto \_ \}\ l :=_{na} v\ \{ l \mapsto v \}$

non-local cross-core communication❓

$\{ l \mapsto_{at} \_ \}\ l :=_{at} v\ \{ \exists V * l \mapsto_{at} (v,V) \}$

$\{ l \mapsto_{at} (v,V) \}\ *^{at} l\ \{ v.\ l \mapsto_{at} (v,V) * \exists V \}$

the local state *temporarily* explicit with *modalities*

Communicating local-state dependent resources

$\{ l \mapsto_{at} \_ \}\ l :=_{at} v\ \{ \exists V * l \mapsto_{at} (v,V) \}$

➡ releasing resources: $\{ P * l \mapsto_{at} \_ \}\ l :=_{at} v\ \{ \exists V * @V\ P * l \mapsto_{at} (v,V) \}$

implicitly local-state dependent

explicitly local-state dependent

$\{ l \mapsto_{at} (v,V) \}\ *^{at}l\ \{ v.\ \ l \mapsto_{at} (v,V) * \exists V \}$

➡ acquiring resources: $\{ @V\ P * l \mapsto_{at} (v,V) \}\ *^{at}l\ \{ v.\ \ l \mapsto_{at} (v,V) * \exists V * P \}$

Some properties

INTRO: P ⊢ ∃ j, ⊒{L}j ∗ @(L,j) P

ELIM: @(L,j) P ⊢ ⊒{L}j -∗ P

COMM: Lj ## Lk → @(Lj,j) @(Lk,k) P ⊣⊢ @(Lk, k) @(Lj, j) P