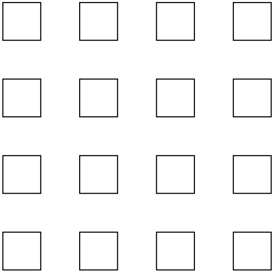
Dynamic X10

Resource-Aware Programming for Higher Efficiency

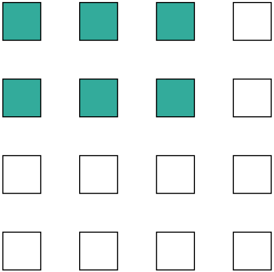
Manuel Mohr, Andreas Zwinkau, Sebastian Buchwald, Matthias Braun

Chair for Programming Paradigms, Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT)

Motivation

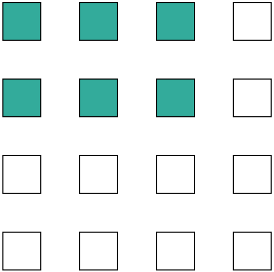


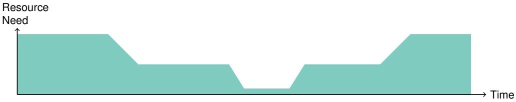
Motivation



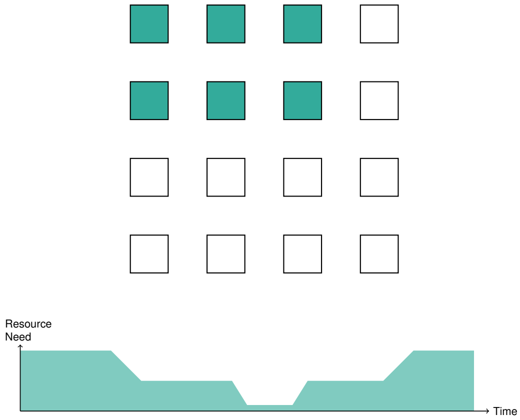
Motivation

Karlsruhe Institute of Technology





Motivation

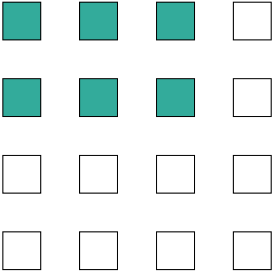


- Static exclusive resource allocation decreases overall efficiency

Dynamic Exclusive Resource Allocation

- Allocate resources exclusively
- But make the allocation dynamically changeable (i.e., at run-time)

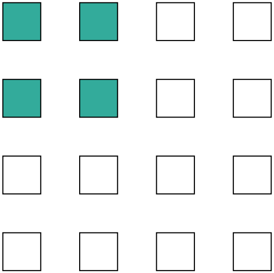
⇒ Resizable claims



Dynamic Exclusive Resource Allocation

- Allocate resources exclusively
- But make the allocation dynamically changeable (i.e., at run-time)

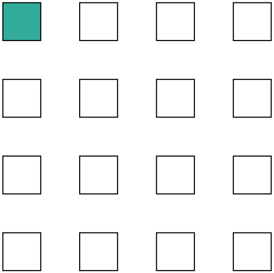
⇒ Resizable claims

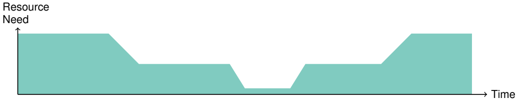


Dynamic Exclusive Resource Allocation

- Allocate resources exclusively
- But make the allocation dynamically changeable (i.e., at run-time)

⇒ Resizable claims

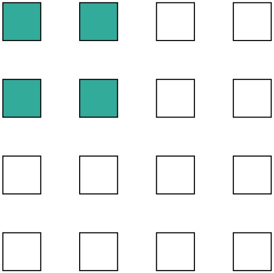


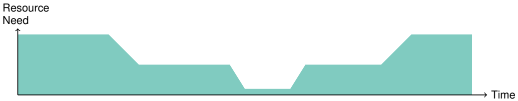


Dynamic Exclusive Resource Allocation

- Allocate resources exclusively
- But make the allocation dynamically changeable (i.e., at run-time)

⇒ Resizable claims

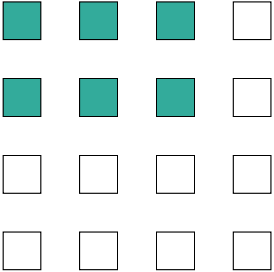




Dynamic Exclusive Resource Allocation

- Allocate resources exclusively
- But make the allocation dynamically changeable (i.e., at run-time)

⇒ Resizable claims

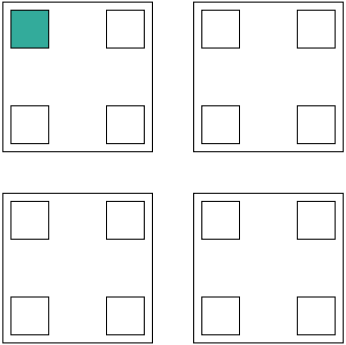


Scope of This Talk

Dynamic X10: framework for programming with resizable claims

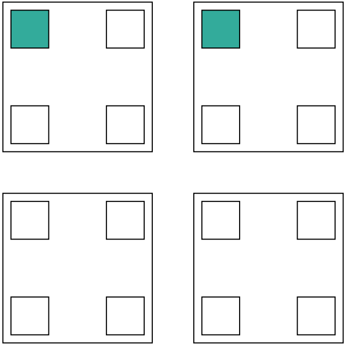
- Integration into existing X10: what needs to change?
- Resource management for improving global efficiency
- Brief evaluation of programming effort for developers

Claim Concept and X10



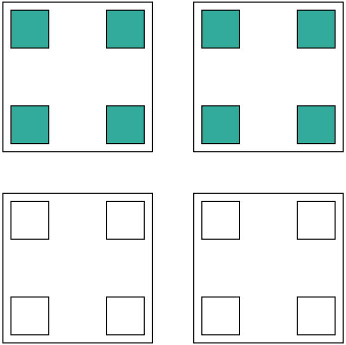
- Claim: set of cores with exclusive access
  - In general: different kinds of resources
- Application's view is restricted to resources in its claim

Claim Concept and X10



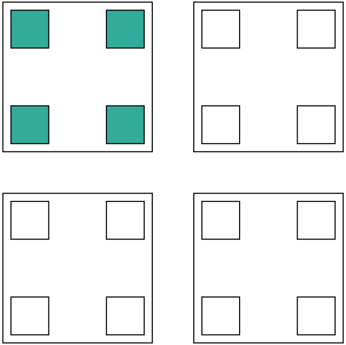
- Claims can be resized with method `reinvade()`
- Claims can span multiple shared-memory domains
  $\Rightarrow$ can contain multiple places

Claim Concept and X10



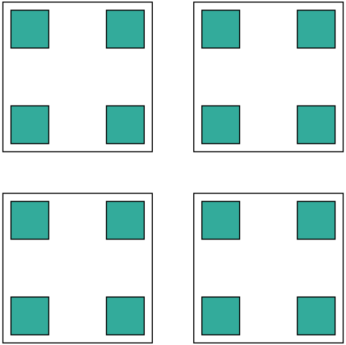
- Claims can be resized with method `reinvade()`
- Claims can span multiple shared-memory domains
  $\Rightarrow$ can contain multiple places

Claim Concept and X10



- Claims can be resized with method `reinvade()`
- Claims can span multiple shared-memory domains
  $\Rightarrow$ can contain multiple places
- Dynamic X10: places can *appear* and *disappear* at runtime

Claim Concept and X10

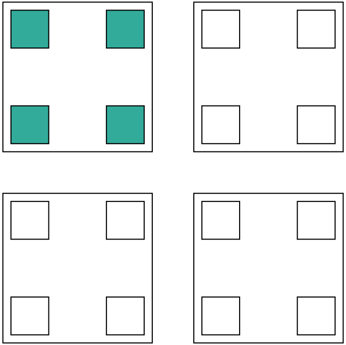


- Natural Embedding: Traditional X10 program behaves like Dynamic X10 program running inside claim containing all cores

Isolation using Multiple Claims

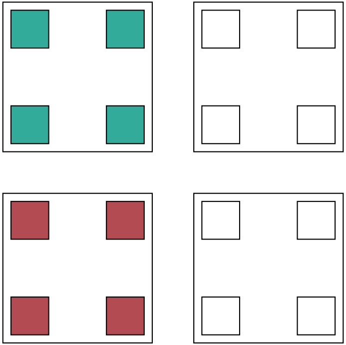
- Alternative to resizing existing claim: creating a *new* claim
- Useful property: isolation
  - Concurrent resizing requests using `reinvade()` $\Rightarrow$ bad
- Method to create and destroy claims: `invade()` and `retreat()`

Multiple Claims and X10

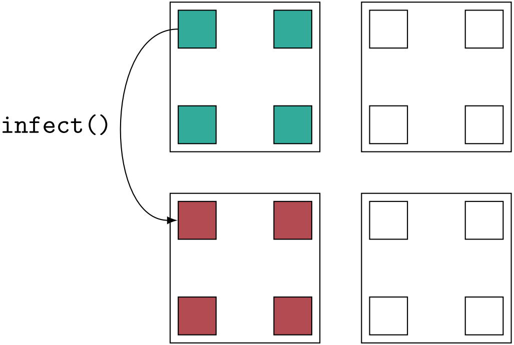


Multiple Claims and X10

Karlsruhe Institute of Technology



Multiple Claims and X10



`infect()` ≙ `at` on claim-level

- Closure as argument (with deep copy semantics)
- Closure runs *inside* the new claim
- Navigation inside new claim with `at` and `async`

Example Usage

```
val claim = Claim.getCurrentClaim();
claim.reinvade(...);
at (here.next()) async { /* do work */ };
val newClaim = Claim.invade(...);
newClaim.infect(() => { /* do more work */ });
```

Example Usage

```
val claim = Claim.getCurrentClaim();
claim.reinvade(...);
at (here.next()) async { /* do work */ };
val newClaim = Claim.invade(...);
newClaim.infect(() => { /* do more work */ });
```

Question: Who changes the allocation? And how?

The application

- No information about other claims

⇒ Does not optimize well in multi-application scenario

Example Usage

```
val claim = Claim.getCurrentClaim();
claim.reinvade(...);
at (here.next()) async { /* do work */ };
val newClaim = Claim.invade(...);
newClaim.infect(() => { /* do more work */ });
```

Question: Who changes the allocation? And how?

The application

- No information about other claims

⇒ Does not optimize well in multi-application scenario

The operating system

- Not enough information about applications

⇒ Does not optimize well in multi-application scenario

Constraints

- Idea: pass application-specific information to OS

⇒ `invade()`/`reinvade()` take resource description: *constraints*

- Modeled as class hierarchy in X10, can be combined using `&&` and `||`
- Most important: `PEQuantity` and `ScalabilityCurve`

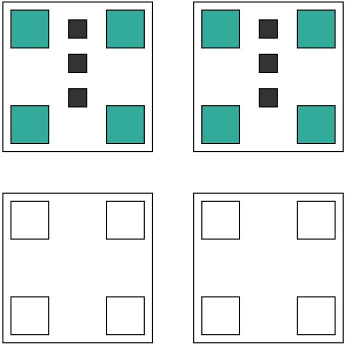
```
Claim.invade(new PEQuantity(1, 3)
             && new ScalabilityCurve([130, 150, 160]));
```

- Scalability information via offline experiments or online monitoring

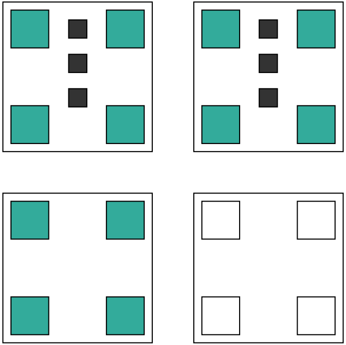
Necessary X10 Adaptions

1. Static fields $\rightarrow$ methods
   - `Place.MAX_PLACES` $\rightarrow$ `Place.maxPlaces()` to query current claim
2. Adapt method implementations
   - `Place.places()` must query current claim
3. Usage of `DistArray`s

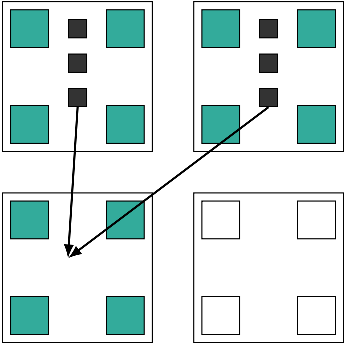
DistArrays



DistArrays

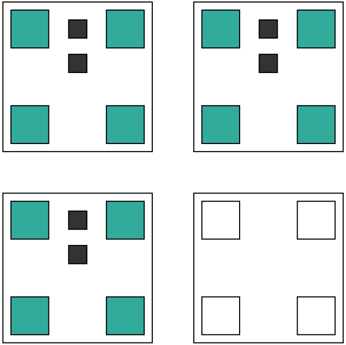


DistArrays



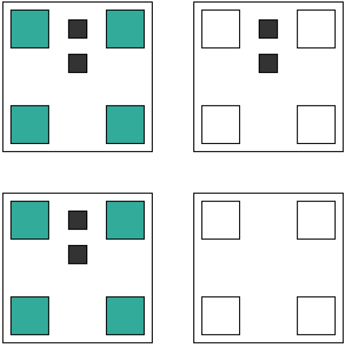
- Create new `DistArray`, move data (library routine)

DistArrays



- Create new `DistArray`, move data (library routine)

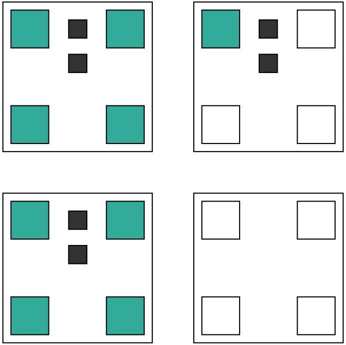
DistArrays



- What if we lose places? ⇒ data inaccessible

DistArrays

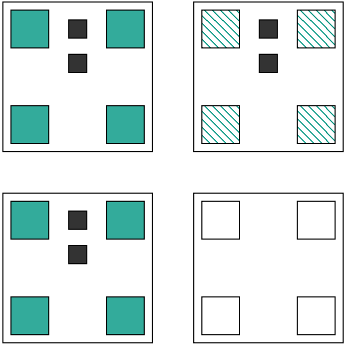
Karlsruhe Institute of Technology



Solution 1: “Sticky claims”

- Forbid losing places
- Simple, but severely restricts reallocation $\Rightarrow$ decreased efficiency

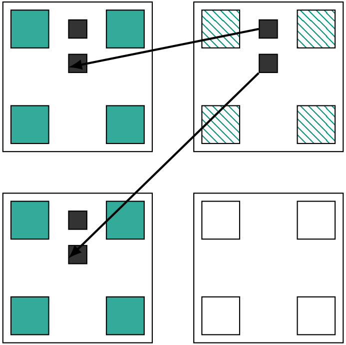
DistArrays



Solution 2: Give opportunity to redistribute

- Requires programmer effort
- More flexible resource reallocation

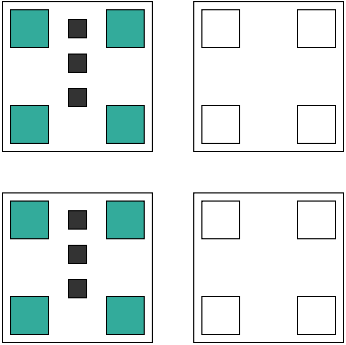
DistArrays



Solution 2: Give opportunity to redistribute

- Requires programmer effort
- More flexible resource reallocation

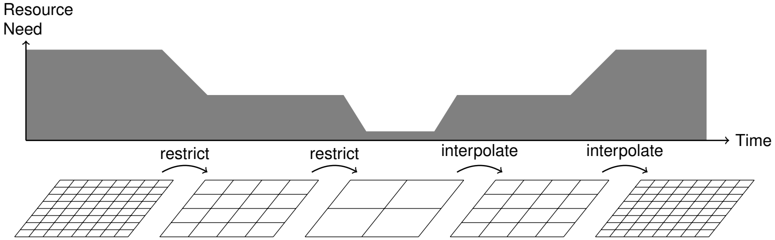
DistArrays



Solution 2: Give opportunity to redistribute

- Requires programmer effort
- More flexible resource reallocation

Preliminary Evaluation: Multigrid



- Prototype implementation of Dynamic X10
- Multigrid: Calls `reinvade()` when resizing grid
  - Uses grid size to compute number of cores
- Prototype uses sticky claims $\Rightarrow$ no disappearing places
- `DistArray` redistribution after `reinvade()`
- $\approx$ 50 lines of additional code compared to traditional X10

Conclusion

We have:

- Integrated claim concept and dynamic exclusive allocation into X10
- Adapted X10 to deal with varying number of places
- Shown that constraints can solve resource management problem
- Implemented a first version of “Dynamic X10”
- Given a preliminary evaluation of programming effort

Backup: Shared Memory Systems

Neither exclusive, nor static resource allocation:

- Resource virtualization: threads instead of CPUs
- Threads multiplexed onto physical cores

Not for free:

- #threads $\gg$ #cores
- Cache issues
- Scheduler has no application-specific knowledge (e.g., barrier synchronization)

Backup: Invasive Computing

- Dynamic X10 implemented in the scope of “Invasive Computing” project
- Looks at future tiled many-core architectures ($>$ 1000 cores)
  - Share characteristics of supercomputers
- Custom operating system
  - Claims as central data structure
  - Claim-aware scheduler