Invasive Malleable Applications

Sebastian Buchwald, Manuel Mohr, **Andreas Zwinkau**

Karlsruhe Institute of Technology

TCRC 89 "Invasive Computing"

ATPS 2015

No faster CPUs, only more of them

More cores means slower cores

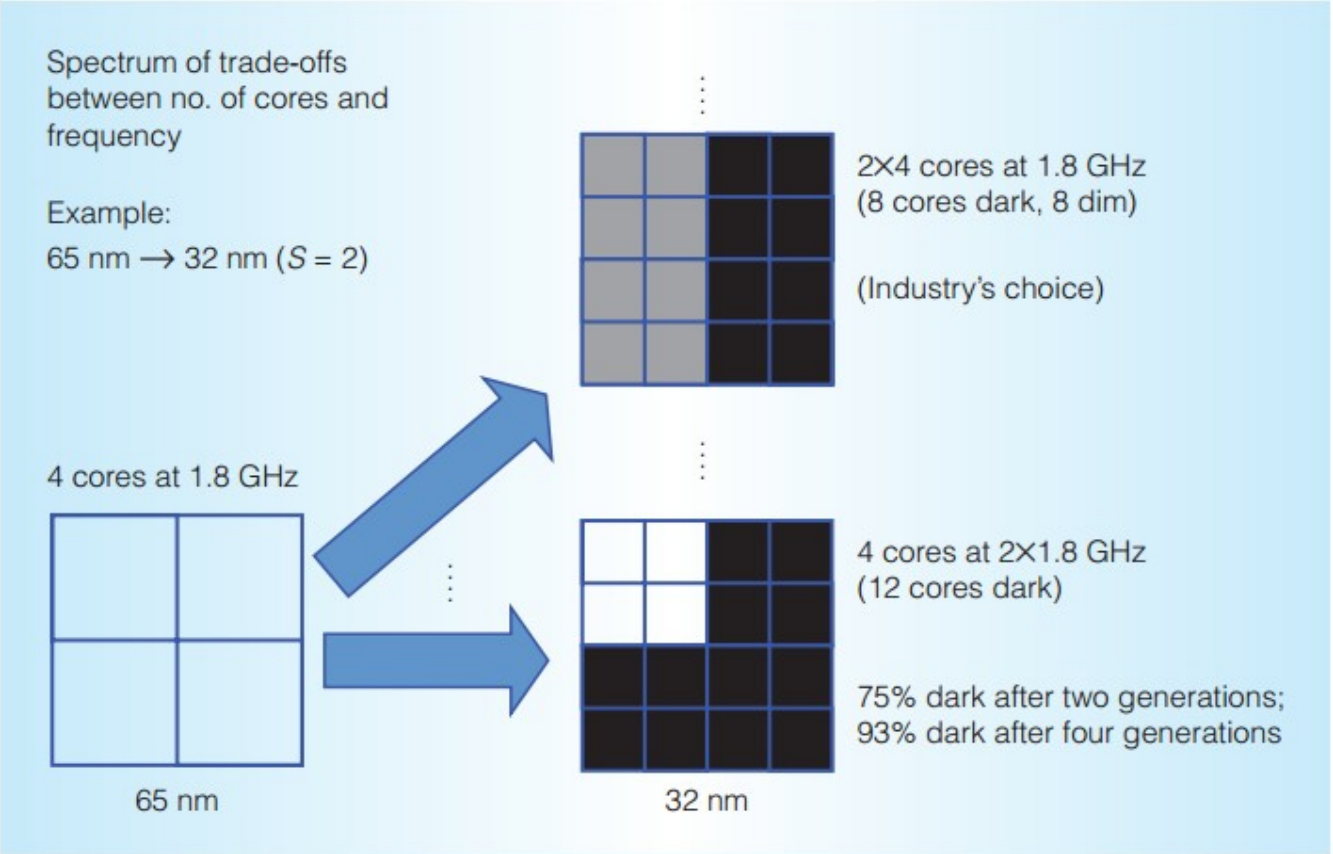
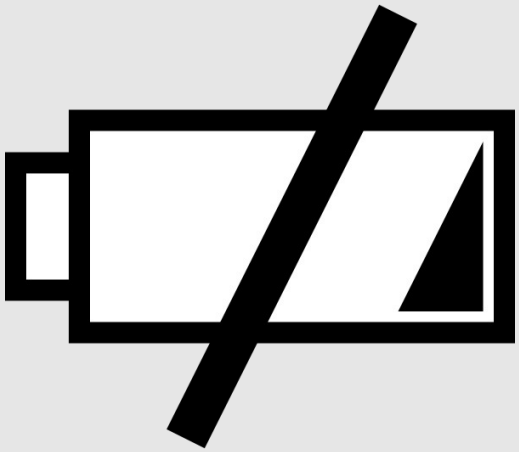


Figure 1. Multicore scaling leads to large amounts of dark silicon.[3] Across two process generations, there is a spectrum of trade-offs between frequency and core count; these include increasing core count by 2× but leaving frequency constant (top), and increasing frequency by 2× but leaving core count constant (bottom). Any of these trade-off points will have large amounts of dark silicon.

M. B. Taylor, **A Landscape of the New Dark Silicon Design Regime**, Micro, IEEE , vol.33, no.5, 2013

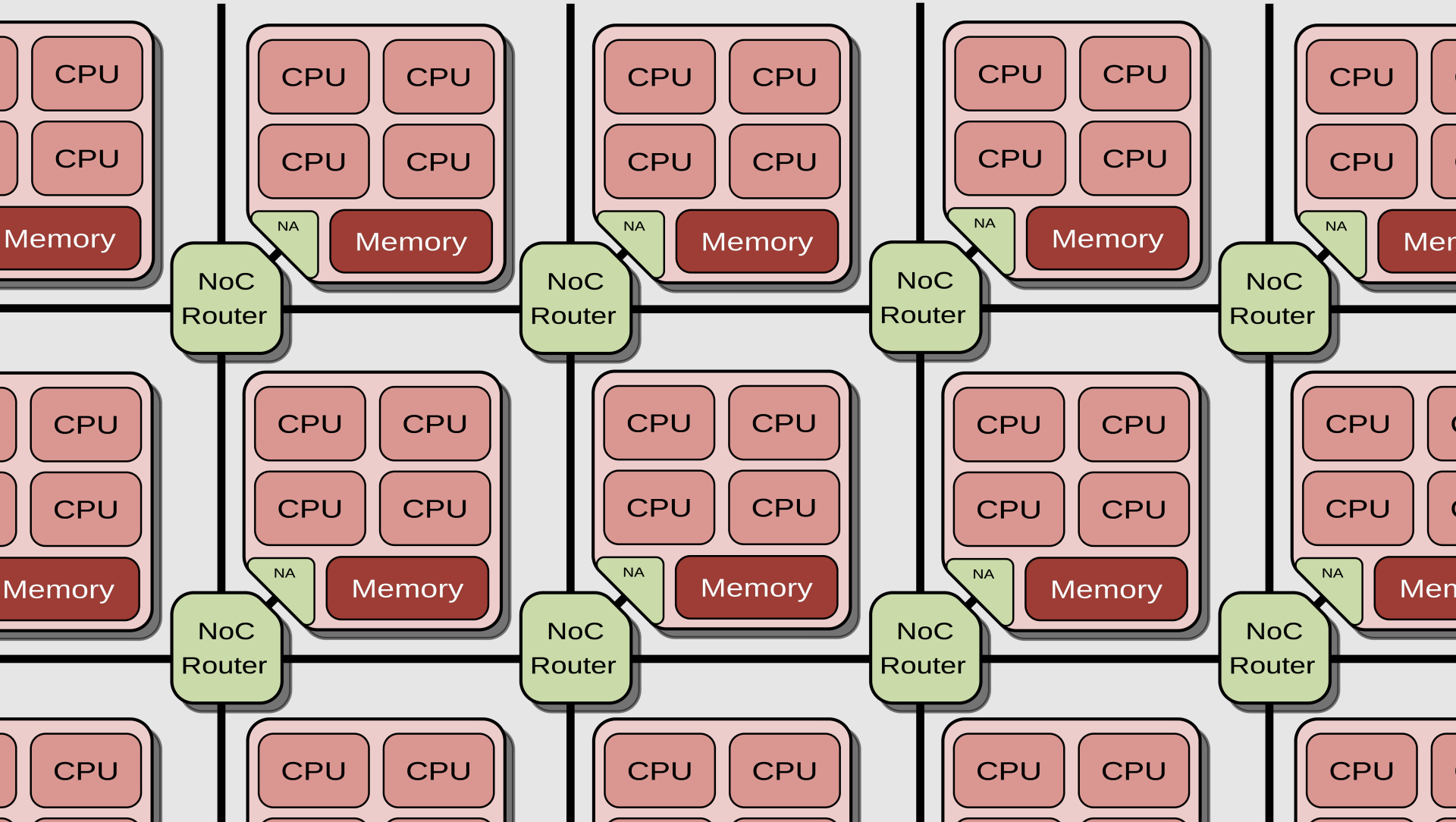
Battery power is more important these days



Challenge:

Use more cores
more **efficiently**

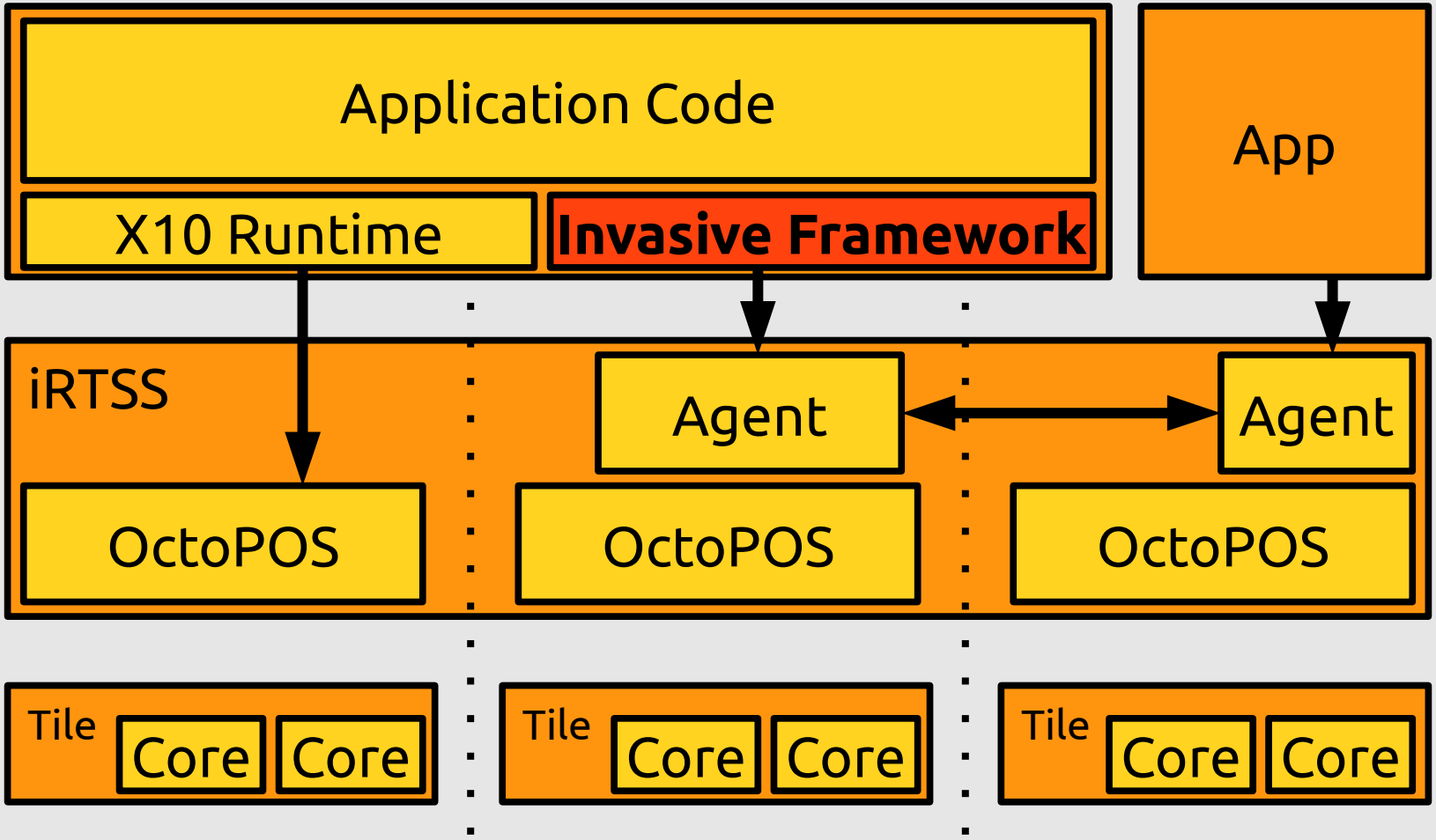
Invasive Computing means **Tiled Manycore Architectures**



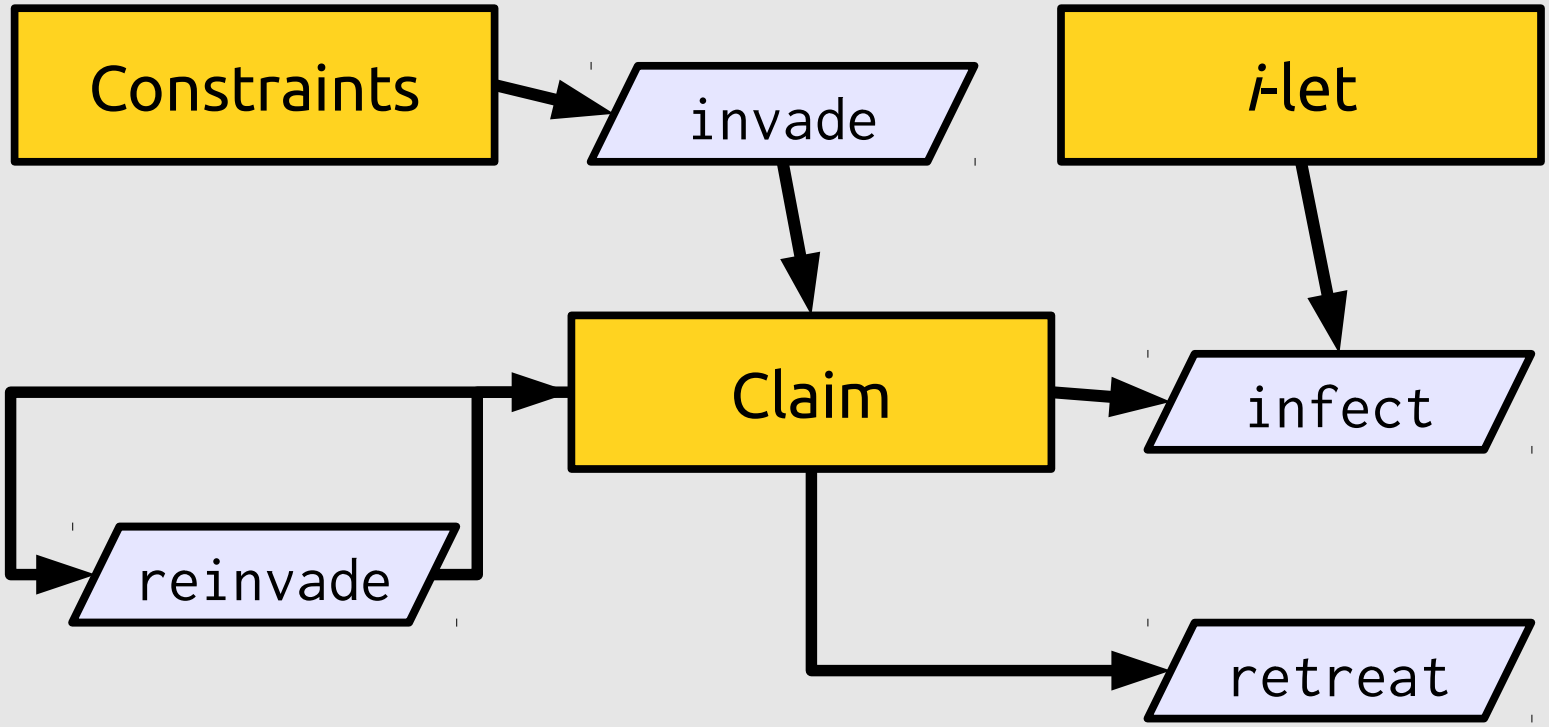
Invasive Computing means **completely rewritten** stack

Custom Hardware	iNoC, CiC, *i*-Core
Custom Operating System	iRTSS, OctoPOS
Custom Programming Language	invadeX10
Applications ported/**rewritten**	HPC, Robotics, ...

The invasive framework gives access to more OS functionality



Invasive Computing is about invade-infect-retreat



Invasive Computing uses X10

```
val ilet = (id:IncarnationID)=>{
  Console.OUT.println("Hello World");
}

val constraints = new PEQuantity(4,10)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
claim.infect(ilet);
claim.retreat();
```

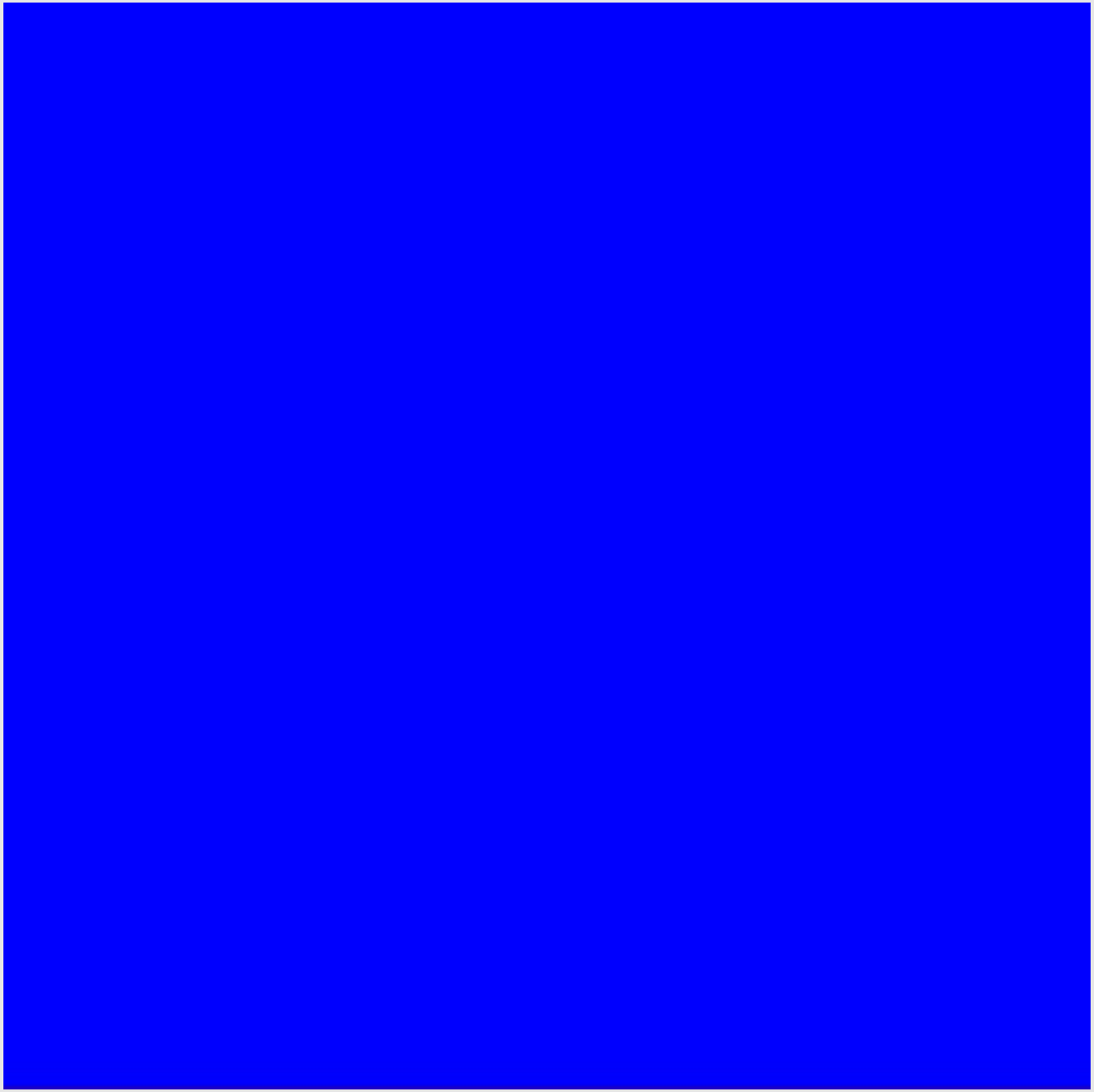
Invasive Applications are **Malleable**

D. G. Feitelson, L. Rudolph; **Towards convergence in job schedulers for parallel supercomputers**, IPPS 1996

	on submission	at runtime
User decides	rigid	evolving
System decides	moldable	**malleable**

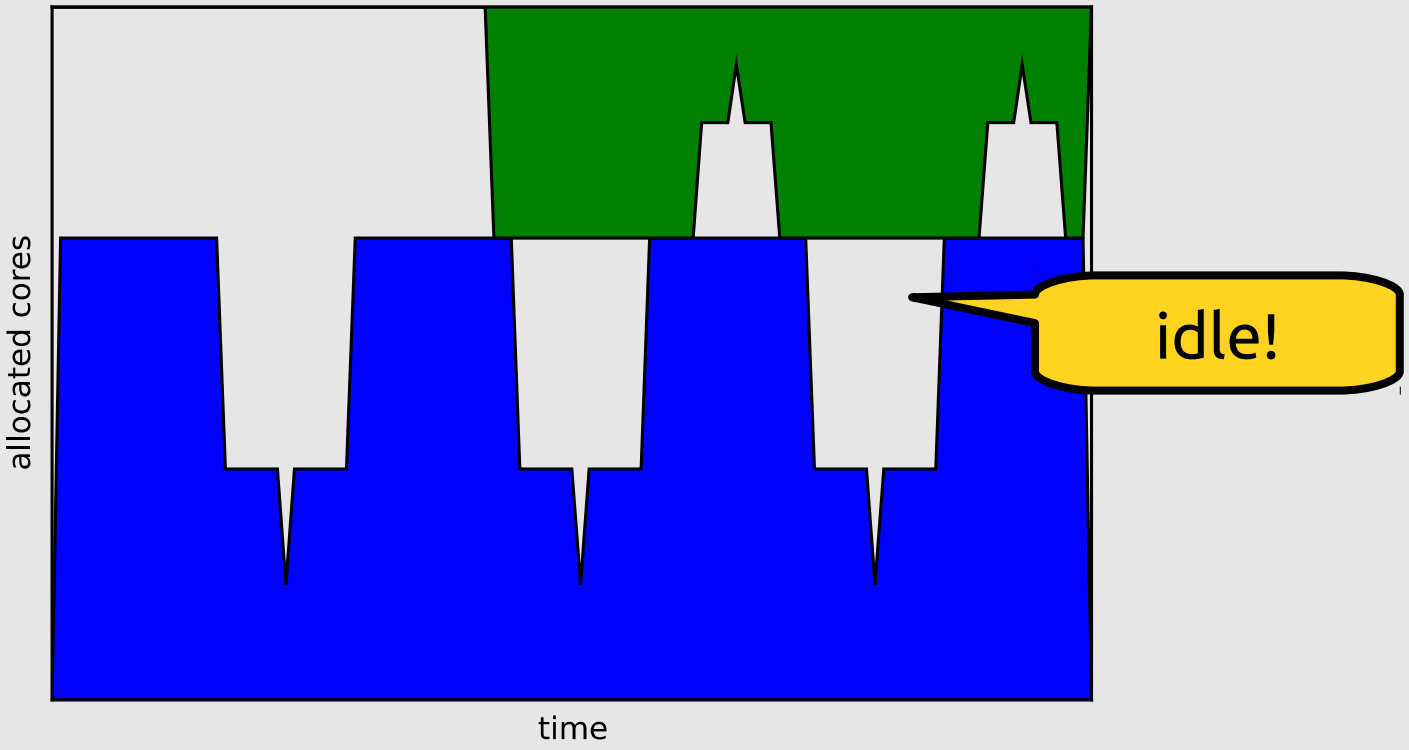
Example:

Heat Dissipation with **Multigrid** Approach

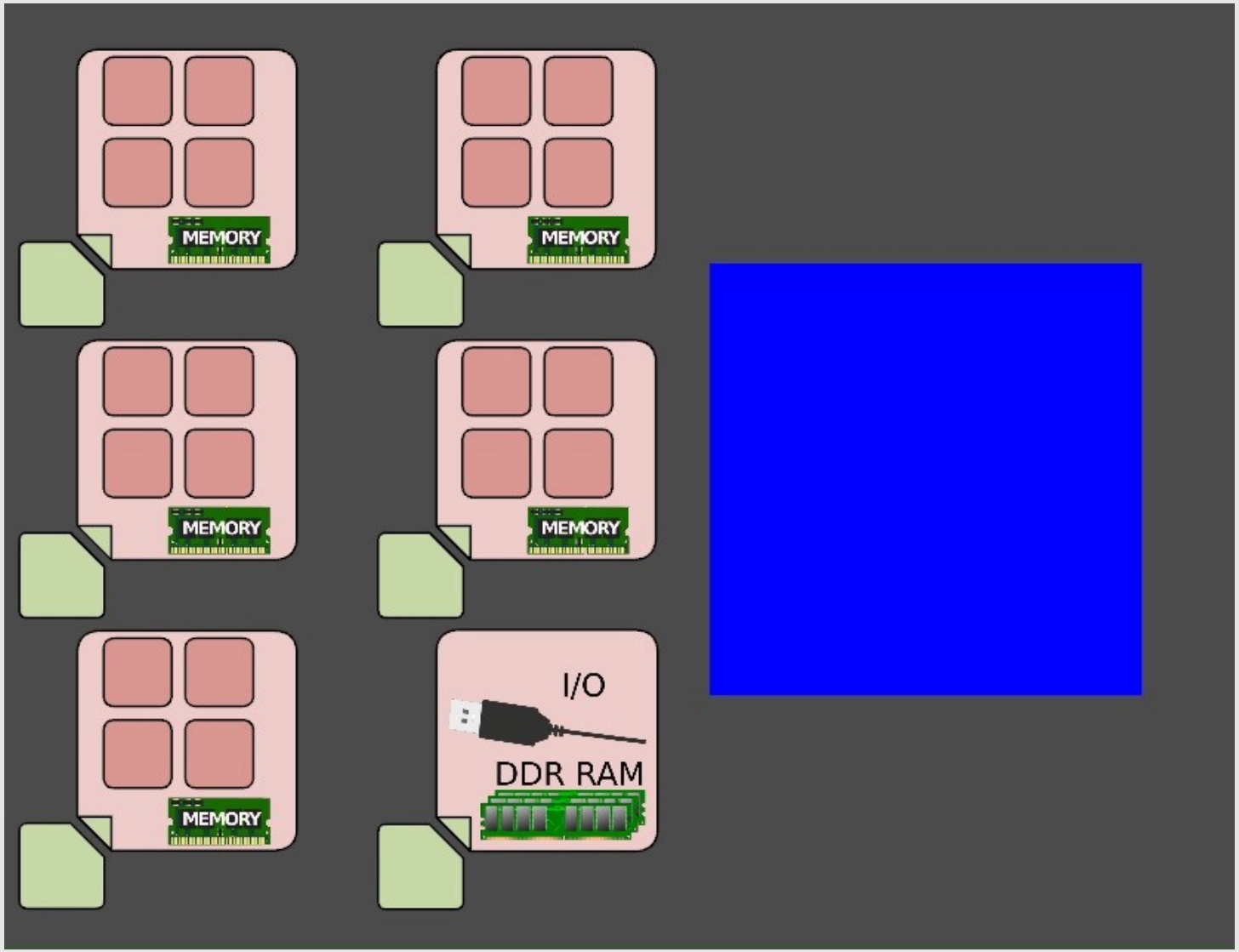


M. Schreiber, A. Zwinkau, et al. **Invasive computing in HPC with X10,** X10 Workshop 2013

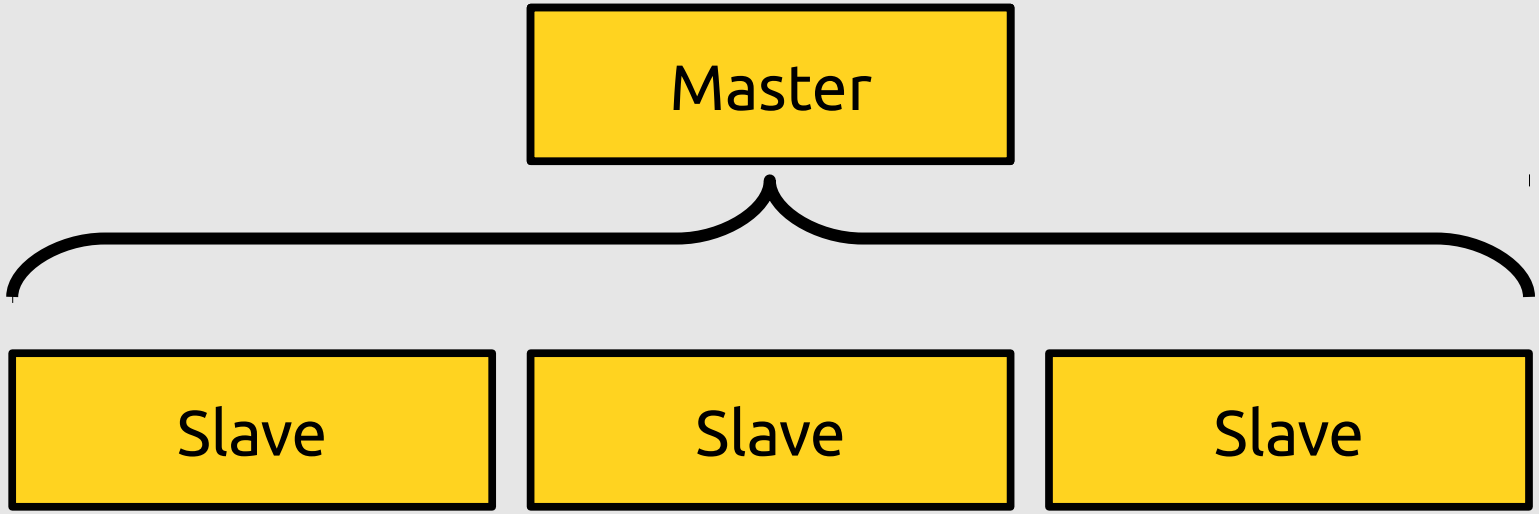
Multigrid lets resource **idle**



Invasive Apps exchange cores

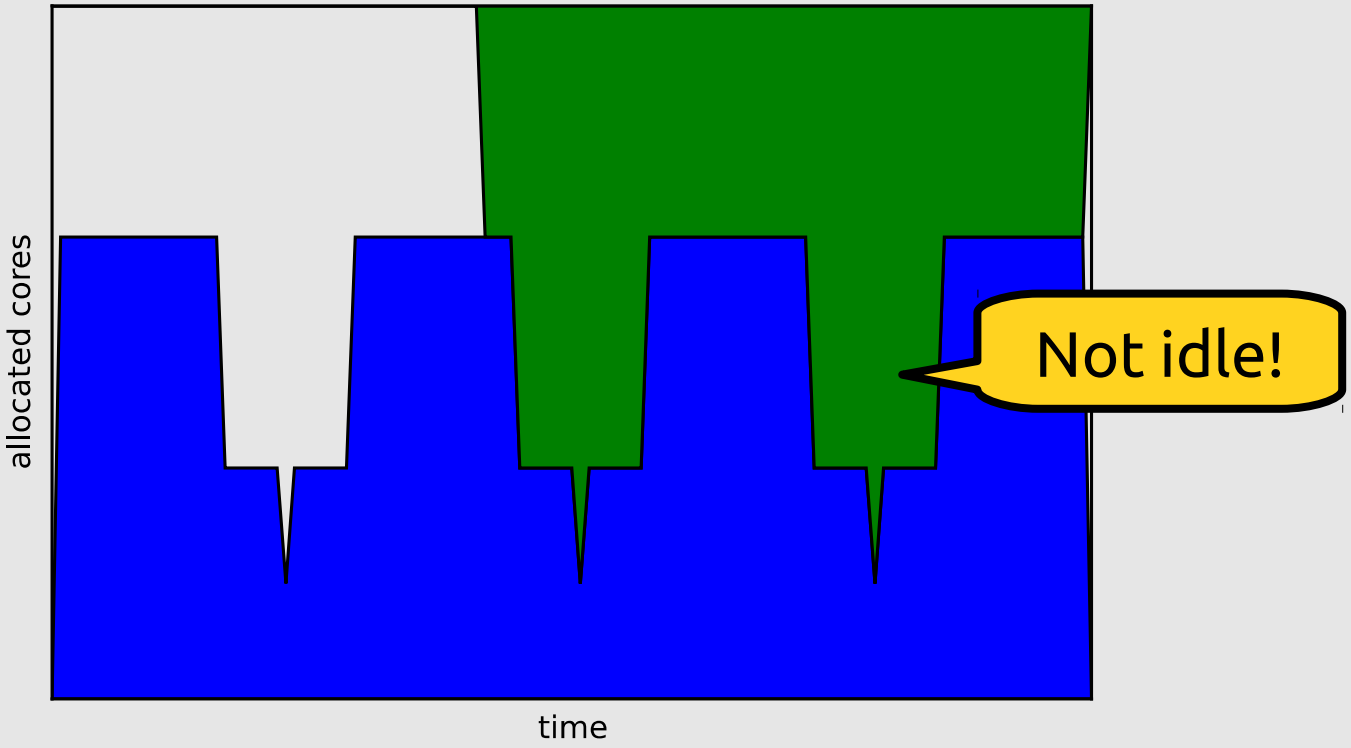


Asynchronously Malleable:
„System decides **any time** at runtime“

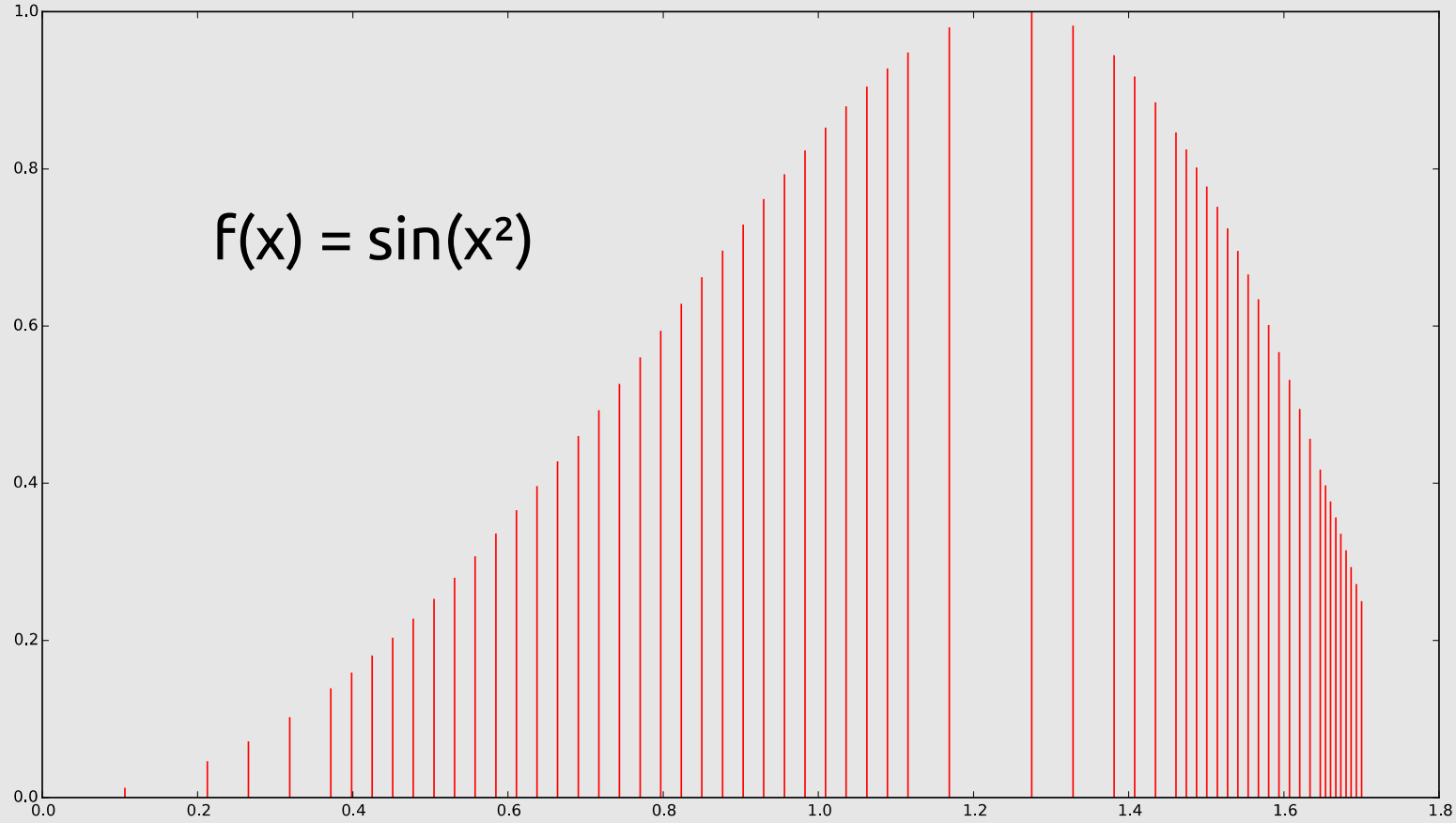


Works nicely for Master-Slave Applications

Asynchronously Malleable is **more efficient**



Integration is async-malleable



Sorting is async-malleable

Patrick Flick, Peter Sanders, Jochen Speck
Malleable Sorting
IPDPS 2013

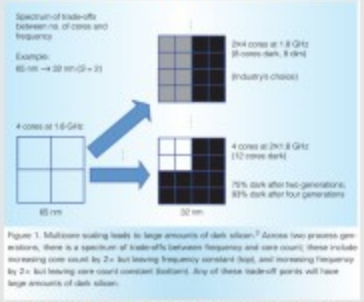
```
val ilet = (id:IncarnationID)=>{
  for (job in queue) {
    if (queue.checkTermination(id)) break;
    job.do();
} }
val resizeHandler = (add:List[PE], remove:List[PE])=>{
  for (pe in add) queue.addWorker(pe,ilet);
  queue.adapt();
  for (pe in remove) queue.signalTermination(pe);
}
val constraints = new PEQuantity(4,10)
  && new AsyncMalleable(resizeHandler)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
queue.adaptTo(claim);
claim.infect(ilet);
claim.retreat();
```

```
val ilet = (id:IncarnationID)=>{
  for (job in queue) {
    if (queue.checkTermination(id)) break;
    job.do();
} }
val resizeHandler = (add:List[PE], remove:List[PE])=>{
  for (pe in add) queue.addWorker(pe,ilet);
  queue.adapt();
  for (pe in remove) queue.signalTermination(pe);
}
val constraints = new PEQuantity(4,10)
  && new AsyncMalleable(resizeHandler)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
queue.adaptTo(claim);
claim.infect(ilet);
claim.retreat();
```

```
val ilet = (id:IncarnationID)=>{
  for (job in queue) {
    if (queue.checkTermination(id)) break;
    job.do();
} }
val resizeHandler = (add:List[PE], remove:List[PE])=>{
  for (pe in add) queue.addWorker(pe,ilet);
  queue.adapt();
  for (pe in remove) queue.signalTermination(pe);
}
val constraints = new PEQuantity(4,10)
  && new Malleable(resizeHandler)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
queue.adaptTo(claim);
claim.infect(ilet);
claim.retreat();
```

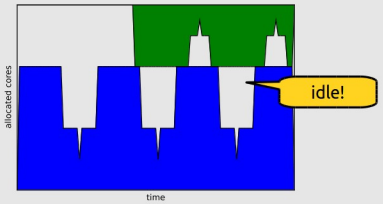
Invasive Malleable Applications

More cores means slower cores

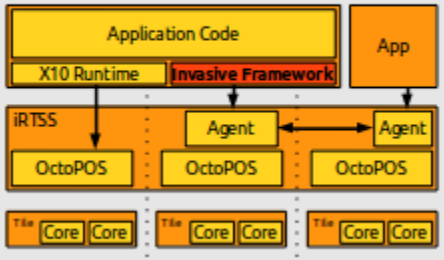


M. B. Taylor, A Landscape of the New Dark Silicon Design Regime, Micro, IEEE, vol.33, no.5, 2013

Multigrid lets resource **idle**

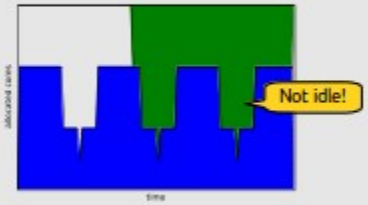


The Invasive Framework gives access to more OS functionality




```
val ilet = (id:IncarnationID)=>{
  for (job in queue) {
    if (queue.checkTermination(id)) break;
    job.do();
} }
val resizeHandler = (add:List[PE], remove:List[PE])=>{
  for (pe in add) queue.addWorker(pe,ilet);
  queue.adapt();
  for (pe in remove) queue.signalTermination(pe);
}
val constraints = new PEQuantity(4,10)
  && new Malleable(resizeHandler)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
queue.adaptTo(claim);
claim.infect(ilet);
claim.retreat();
```

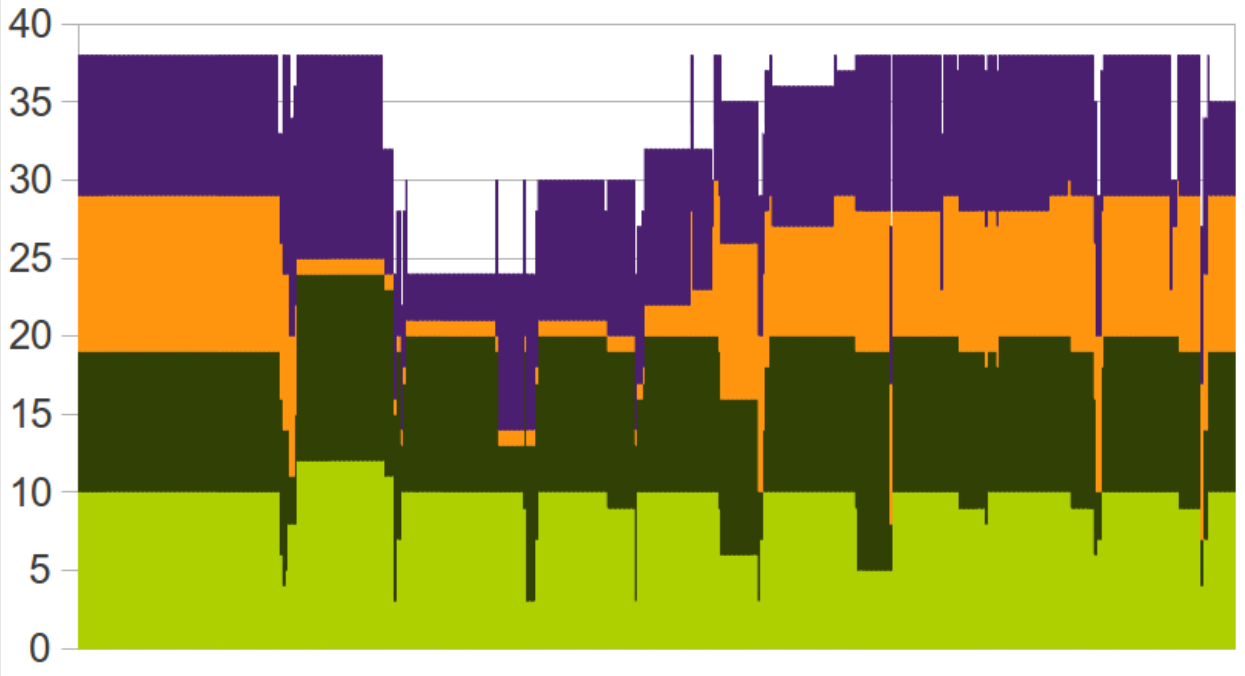
Asynchronously Malleable is **more efficient**



Appendix

Case Study: **Multigrid**

M. Schreiber, A. Zwinkau, et al. **Invasive computing in HPC with X10**
X10 Workshop 2013



```
val ilet = (id:IncarnationID)=>{
  for (job in queue) {
    if (queue.checkTermination(id)) break;
    job.do();
} }
val resizeHandler = (add:List[PE], remove:List[PE])=>{
  for (pe in add) queue.addWorker(pe,ilet);
  queue.adapt();
  for (pe in remove) queue.signalTermination(pe);
}
val constraints = new PEQuantity(4,10)
  && new Malleable(resizeHandler)
  && new ScalabilityHint(speedupCurve);

val claim = Claim.invade(constraints);
queue.adaptTo(claim);
claim.infect(ilet);
claim.retreat();
```