Advanced Charm++ Tutorial

Presented by:

Isaac Dooley & Chao Mei

4/20/2007

PARALLEL PROGRAMMING LAB

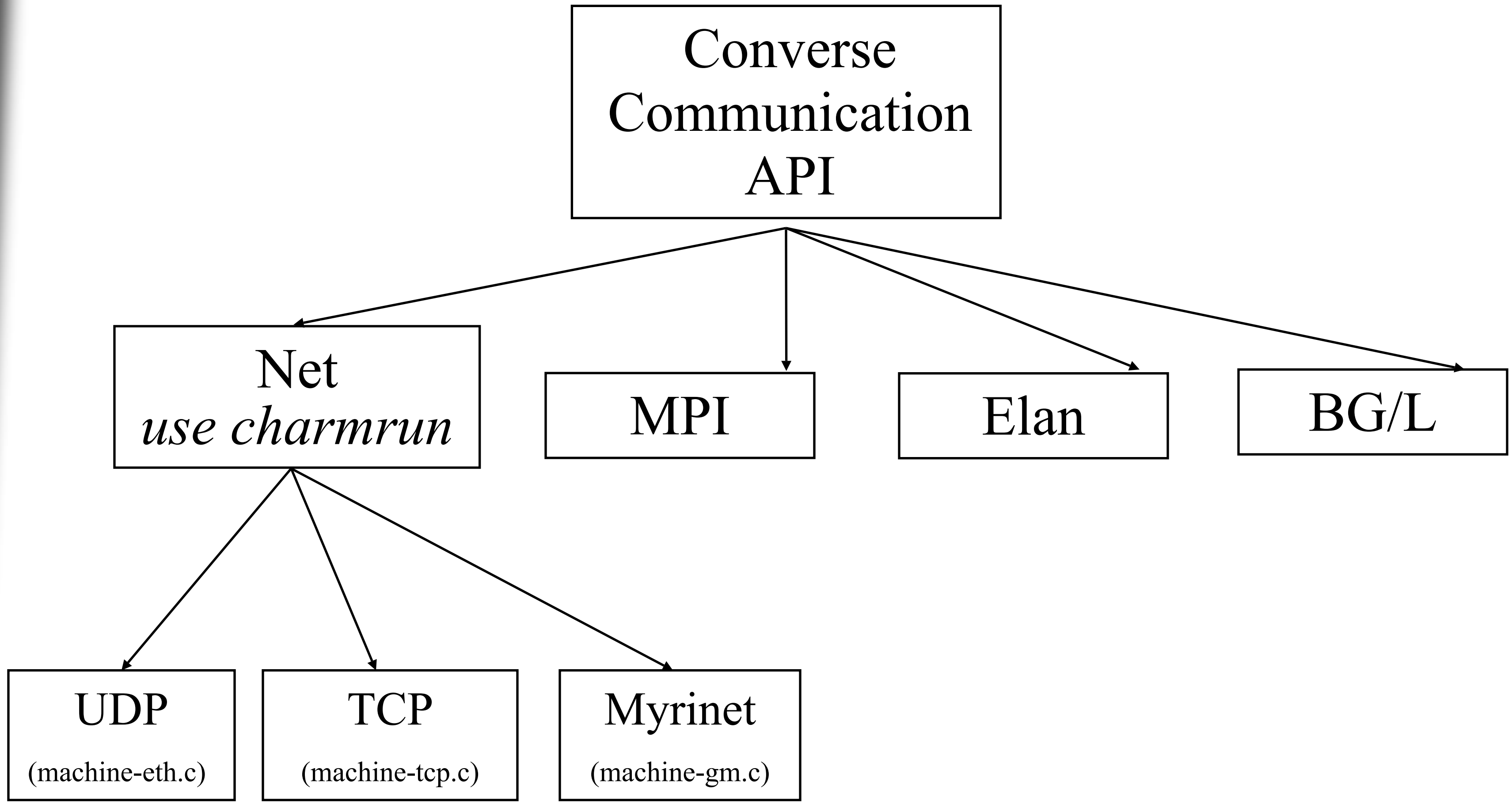
Topics For This Talk

- Building Charm++
- Advanced messaging
- Interface file (.ci)
- Advanced load balancing
- Groups
- Threads
- Delegation
- Array multicast
- SDAG

Charm++ on Parallel Machines

- Runs on:
  - Any machine with MPI, including
    - IBM Blue Gene/L, SP
    - Cray XT3
    - SGI Altix
  - PSC’s Lemieux (Quadrics Elan)
  - Clusters with Ethernet (UDP/TCP)
  - Clusters with Myrinet (GM or MX)
  - Apple clusters
  - Even Windows!
- SMP-Aware (pthreads)

Communication Architecture


Converse Communication API

Net
use charmrun

MPI

Elan

BG/L

UDP
(machine-eth.c)

TCP
(machine-tcp.c)

Myrinet
(machine-gm.c)

Compiling Charm++

```
./build
Usage: build <target> <version> <options> [charmc-options ...]



<target>: converse charm++ LIBS AMPI FEM bigemulator pose jade msa
          doc ps-doc pdf-doc html-doc

  charm++         compile Charm++ core only
  AMPI            compile Adaptive MPI on top of Charm++
  FEM             compile FEM framework
  LIBS            compile additional parallel libraries with Charm++ core
  bigemulator     build additional BigSim libraries
  pose            build POSE parallel discrete event simulator
  jade            build Jade compiler (auto-builds charm++, msa)
  msa             build Multiphase Shared Arrays(MSA) library
```

Compiling Charm++

./build

Usage: build <target> **<version>** <options> [charmc-options ...]

<version>: Basic configurations

```
bluegenel
elan-axp
elan-linux-ia64
exemplar
mpi-axp
mpi-bluegenel
mpi-crayx1
mpi-crayxt3
mpi-exemplar
mpi-hp-ia64
mpi-linux
mpi-linux-amd64
mpi-linux-axp
mpi-linux-ia64
mpi-origin
mpi-ppc-darwin
mpi-sol
mpi-sol-amd64
mpi-sp
ncube2
net-axp
net-cygwin
net-darwin-x86
net-hp
net-hp-ia64
net-irix
net-linux
net-linux-amd64
net-linux-axp
net-linux-cell
net-linux-ia64
net-linux-ppc
net-ppc-darwin
net-rs6k
net-sol
net-sol-amd64
net-sol-x86
net-sun
net-win32
net-win64
origin-pthreads
origin2000
portals-crayxt3
shmem-axp
sim-linux
sp3
t3e
uth-linux
uth-win32
vmi-linux
vmi-linux-amd64
vmi-linux-ia64
```

Compiling Charm++

```
./build
Usage: build <target> <version> <options> [charmc-options ...]
```

<version>: Basic configurations

bluegenel
elan-axp
elan-linux-ia64
exemplar
mpi-axp
mpi-bluegenel
mpi-crayx1
mpi-crayxt3
mpi-exemplar
mpi-hp-ia64
mpi-linux
mpi-linux-amd64
mpi-linux-axp
mpi-linux-ia64
mpi-origin
mpi-ppc-darwin
mpi-sol
mpi-sol-amd64
mpi-sp
ncube2
net-axp
net-cygwin
net-darwin-x86
net-hp
net-hp-ia64
net-irix
net-linux
net-linux-amd64
net-linux-axp
net-linux-cell
net-linux-ia64
net-linux-ppc
net-ppc-darwin
net-rs6k
net-sol
net-sol-amd64
net-sol-x86
net-sun
net-win32
net-win64
origin-pthreads
origin2000
portals-crayxt3
shmem-axp
sim-linux
sp3
t3e
uth-linux
uth-win32
vmi-linux
vmi-linux-amd64
vmi-linux-ia64

Compiling Charm++

./build

Usage: build <target> <version> **<options>** [charmc-options ...]

<options>: compiler and platform specific options

Platform specific options (choose multiple if they apply)**:**

```
 lam              Use LAM MPI
 smp              support for SMP, multithreaded charm on each node
 mpt              use SGI Message Passing Toolkit (only for mpi version)
 gm               use Myrinet for communication
 tcp              use TCP sockets for communication (ony for net version)
 vmi              use NCSA's VMI for communication (only for mpi version)
 scyld            compile for Scyld Beowulf cluster based on bproc
 clustermatic     compile for Clustermatic (support version 3 and 4)
 pthreads         compile with pthreads Converse threads
```

Compiling Charm++

./build

Usage: build <target> <version> **<options>** [charmc-options ...]

<options>: compiler and platform specific options

Advanced options:

```
  bigemulator     compile for BigSim simulator
  ooc             compile with out of core support
  syncft          compile with Charm++ fault tolerance support
  papi            compile with PAPI performance counter support (if any)
```

Charm++ dynamic libraries:

```
  --build-shared     build Charm++ dynamic libraries (.so) (default)
  --no-build-shared  don't build Charm++'s shared libraries
```

Compiling Charm++

./build

Usage: build <target> <version> **<options>** [charmc-options ...]

<options>: compiler and platform specific options

Choose a C++ compiler (only one option is allowed from this section):

```
  cc, cc64        For Sun WorkShop C++ 32/64 bit compilers
  cxx             DIGITAL C++ compiler (DEC Alpha)
  kcc             KAI C++ compiler
  pgcc            Portland Group's C++ compiler
  acc             HP aCC compiler
  icc             Intel C/C++ compiler for Linux IA32
  ecc             Intel C/C++ compiler for Linux IA64
  gcc3            use gcc3 - GNU GCC/G++ version 3
  gcc4            use gcc4 - GNU GCC/G++ version 4 (only mpi-crayxt3)
  mpcc            SUN Solaris C++ compiler for MPI
  pathscale       use pathscale compiler suite
```

Compiling Charm++

./build
Usage: build <target> <version> **<options>** [charmc-options ...]

<options>: compiler and platform specific options

Choose a fortran compiler (only one option is allowed from this section):

```
  g95               G95 at http://ww.g95.org
  absoft            Absoft fortran compiler
  pgf90             Portland Group's Fortran compiler
  ifc               Intel Fortran compiler (older versions)
  ifort             Intel Fortran compiler (newer versions)
```

Compiling Charm++

./build

Usage: build <target> <version> <options> **[charmc-options ...]**

<charmc-options>: normal compiler options

```
    -g -O -save -verbose
```

To see the latest versions of these lists or to get more detailed help, run

```
    ./build --help
```

Build Script

- **Build script does:**

  `./build <target> <version> <options> [charmc-options ...]`

  - Creates directories `<version>` and `<version>/tmp`
  - Copies `src/scripts/Makefile` into `<version>/tmp`
  - Does a `"make <target> <version> OPTS=<charmc-options>"` in `<version>/tmp`
- **That's all build does. The rest is handled by the Makefile.**

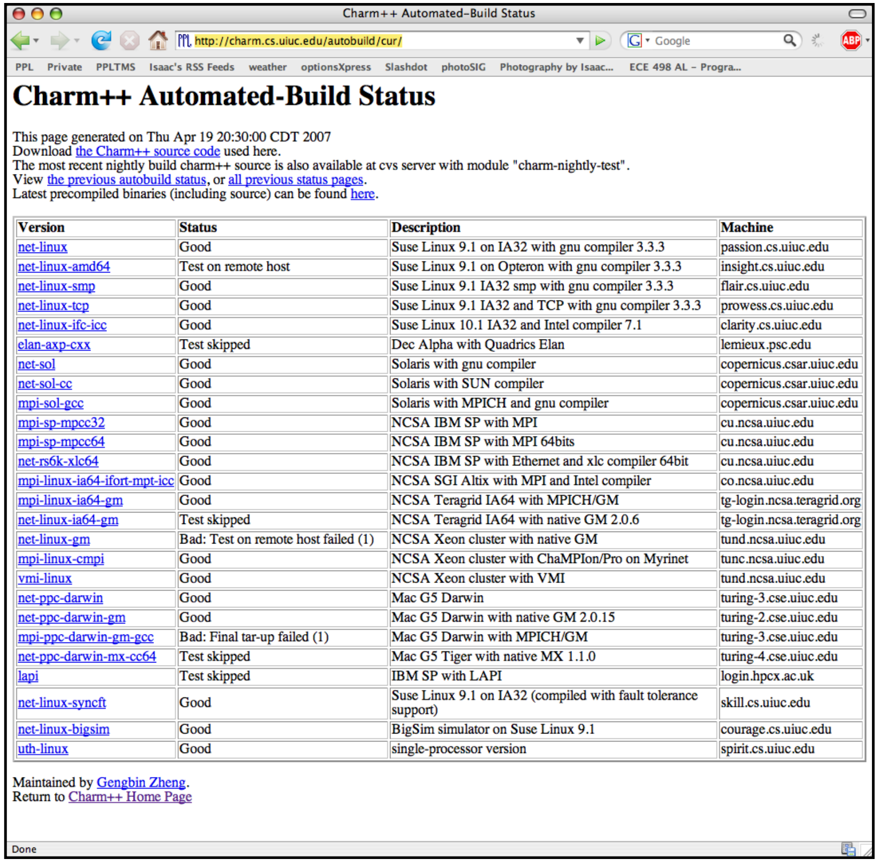
How ‘build’ works

- build AMPI net-linux gm kcc
  - Mkdir net-linux-gm-kcc
  - Cat conv-mach-[kcc|gm|smp].h to conv-mach-opt.h
  - Cat conv-mach-[kcc|gm].sh to conv-mach-opt.sh
  - Gather files from net, etc (Makefile)
  - Make charm++ under
    - net-linux-gm/tmp

What if build fails?

- Use latest version from CVS
- Check the nightly auto-build tests: http://charm.cs.uiuc.edu/autobuild/cur/

- Email: ppl@cs.uiuc.edu



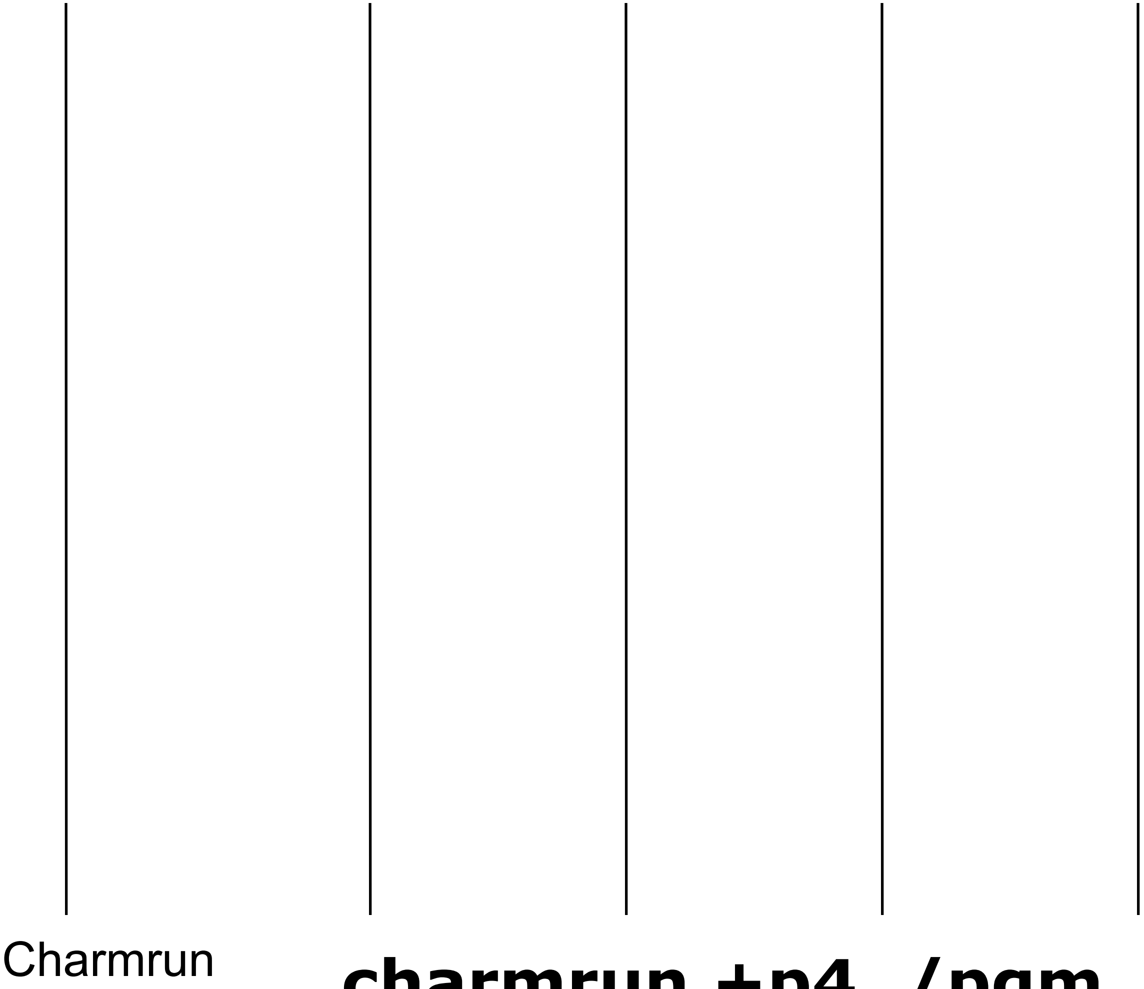
Charm++ Automated-Build Status

This page generated on Thu Apr 19 20:30:00 CDT 2007
Download the Charm++ source code used here.
The most recent nightly build charm++ source is also available at cvs server with module "charm-nightly-test".
View the previous autobuild status, or all previous status pages.
Latest precompiled binaries (including source) can be found here.

Version	Status	Description	Machine
net-linux	Good	Suse Linux 9.1 on IA32 with gnu compiler 3.3.3	passion.cs.uiuc.edu
net-linux-amd64	Test on remote host	Suse Linux 9.1 on Opteron with gnu compiler 3.3.3	insight.cs.uiuc.edu
net-linux-smp	Good	Suse Linux 9.1 IA32 smp with gnu compiler 3.3.3	flair.cs.uiuc.edu
net-linux-tcp	Good	Suse Linux 9.1 IA32 and TCP with gnu compiler 3.3.3	prowess.cs.uiuc.edu
net-linux-ifc-icc	Good	Suse Linux 10.1 IA32 and Intel compiler 7.1	clarity.cs.uiuc.edu
elan-axp-cxx	Test skipped	Dec Alpha with Quadrics Elan	lemieux.psc.edu
net-sol	Good	Solaris with gnu compiler	copernicus.csar.uiuc.edu
net-sol-cc	Good	Solaris with SUN compiler	copernicus.csar.uiuc.edu
mpi-sol-gcc	Good	Solaris with MPICH and gnu compiler	copernicus.csar.uiuc.edu
mpi-sp-mpcc32	Good	NCSA IBM SP with MPI	cu.ncsa.uiuc.edu
mpi-sp-mpcc64	Good	NCSA IBM SP with MPI 64bits	cu.ncsa.uiuc.edu
net-rs6k-xlc64	Good	NCSA IBM SP with Ethernet and xlc compiler 64bit	cu.ncsa.uiuc.edu
mpi-linux-ia64-ifort-mpt-icc	Good	NCSA SGI Altix with MPI and Intel compiler	co.ncsa.uiuc.edu
mpi-linux-ia64-gm	Good	NCSA Teragrid IA64 with MPICH/GM	tg-login.ncsa.teragrid.org
net-linux-ia64-gm	Test skipped	NCSA Teragrid IA64 with native GM 2.0.6	tg-login.ncsa.teragrid.org
net-linux-gm	Bad: Test on remote host failed (1)	NCSA Xeon cluster with native GM	tund.ncsa.uiuc.edu
mpi-linux-cmpi	Good	NCSA Xeon cluster with ChaMPIon/Pro on Myrinet	tunc.ncsa.uiuc.edu
vmi-linux	Good	NCSA Xeon cluster with VMI	tund.ncsa.uiuc.edu
net-ppc-darwin	Good	Mac G5 Darwin	turing-3.cse.uiuc.edu
net-ppc-darwin-gm	Good	Mac G5 Darwin with native GM 2.0.15	turing-2.cse.uiuc.edu
mpi-ppc-darwin-gm-gcc	Bad: Final tar-up failed (1)	Mac G5 Darwin with MPICH/GM	turing-3.cse.uiuc.edu
net-ppc-darwin-mx-cc64	Test skipped	Mac G5 Tiger with native MX 1.1.0	turing-4.cse.uiuc.edu
lapi	Test skipped	IBM SP with LAPI	login.hpcx.ac.uk
net-linux-syncft	Good	Suse Linux 9.1 on IA32 (compiled with fault tolerance support)	skill.cs.uiuc.edu
net-linux-bigsim	Good	BigSim simulator on Suse Linux 9.1	courage.cs.uiuc.edu
uth-linux	Good	single-processor version	spirit.cs.uiuc.edu

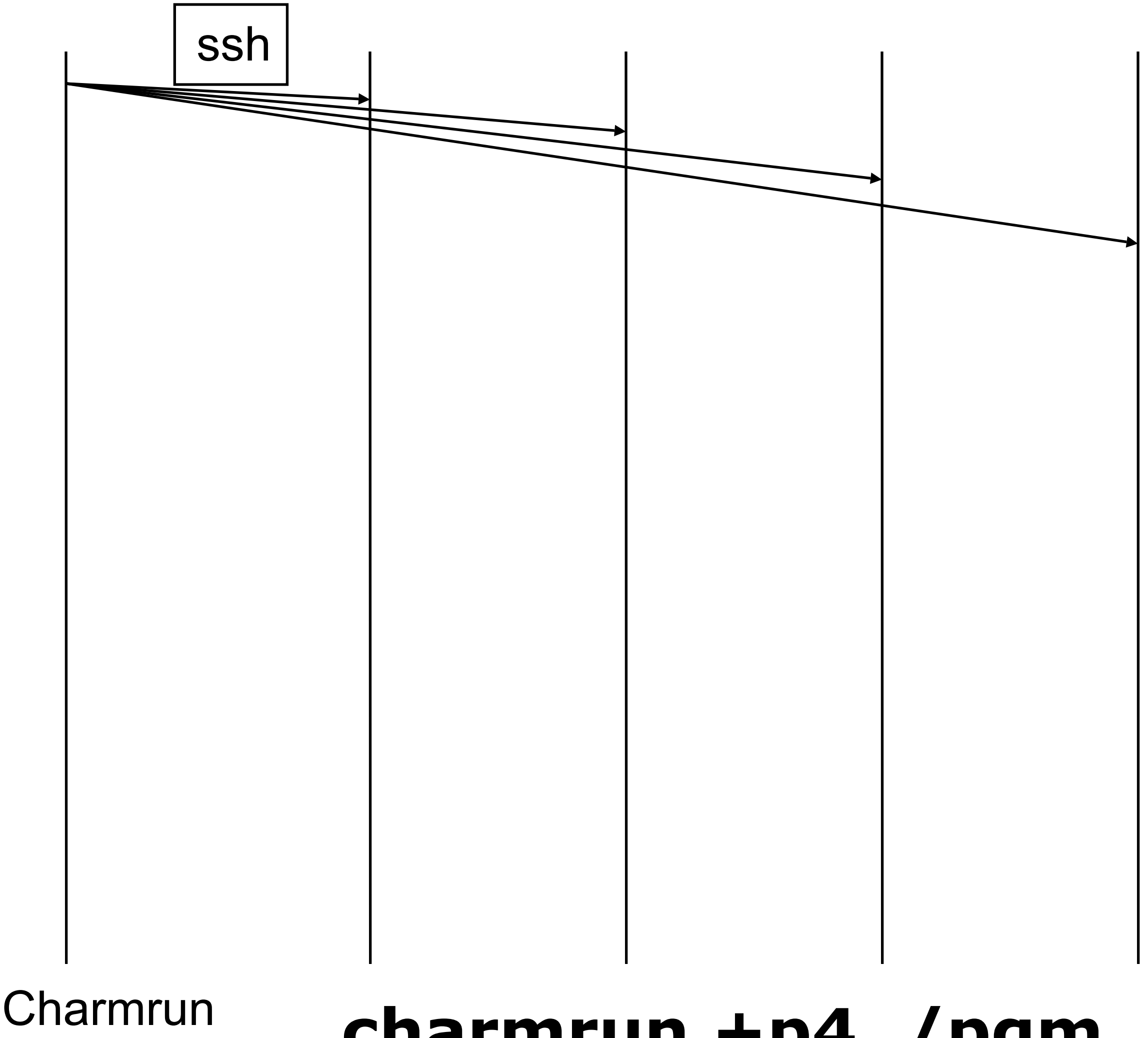
Maintained by Gengbin Zheng.
Return to Charm++ Home Page

How Charmrun Works?



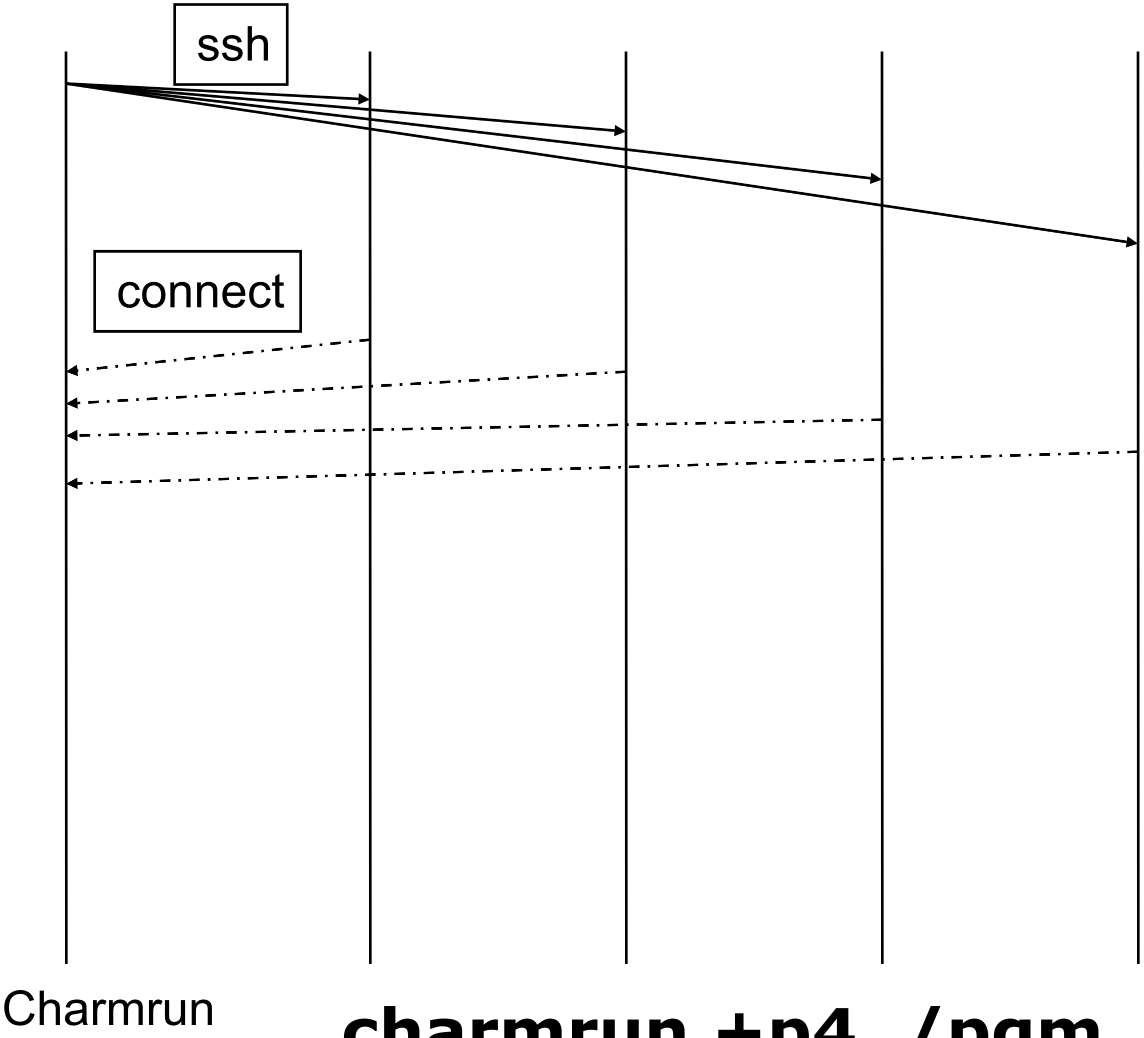
```
charmrun +p4 ./pgm
```

How Charmrun Works?



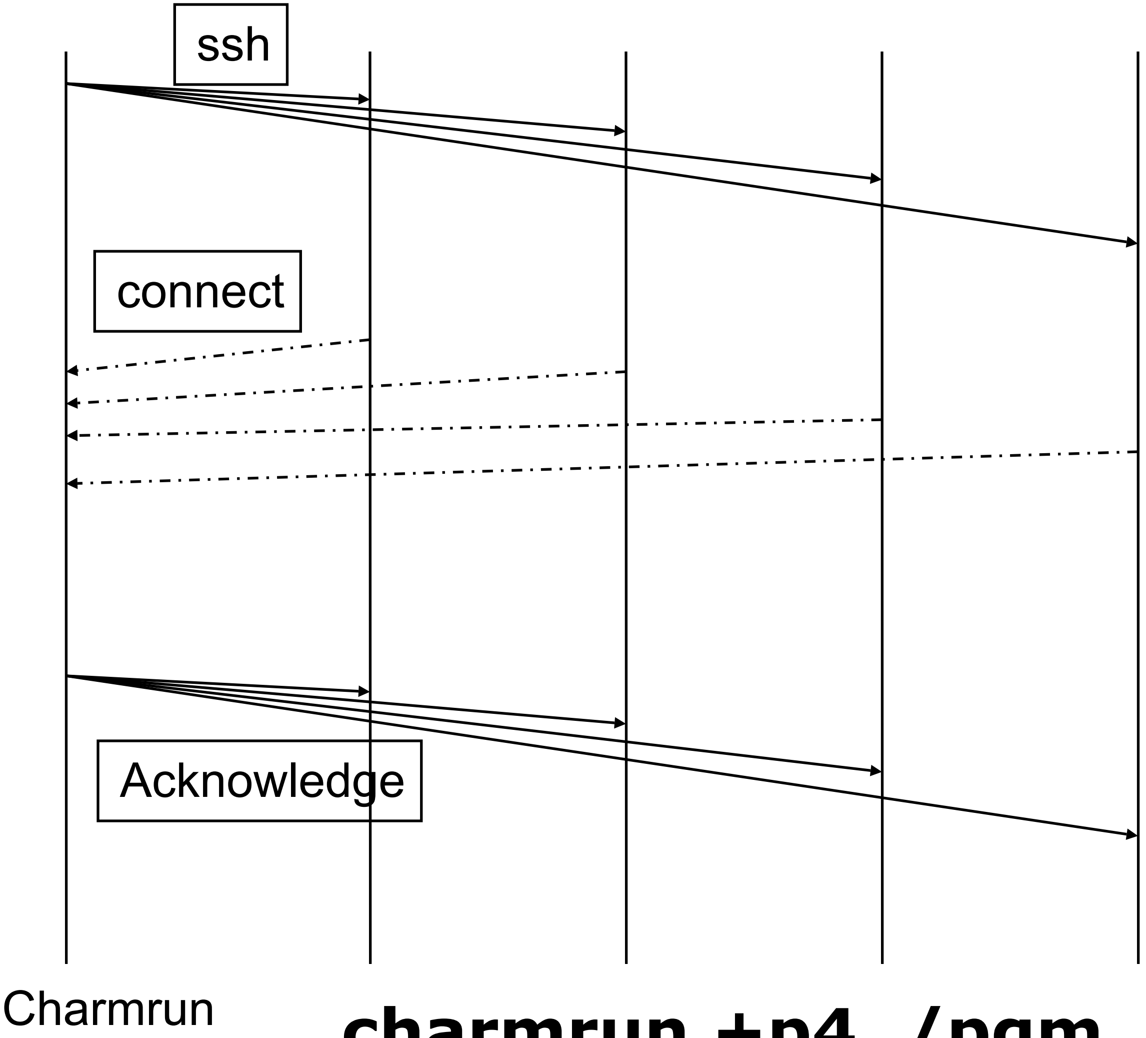
charmrun +p4 ./pgm

How Charmrun Works?



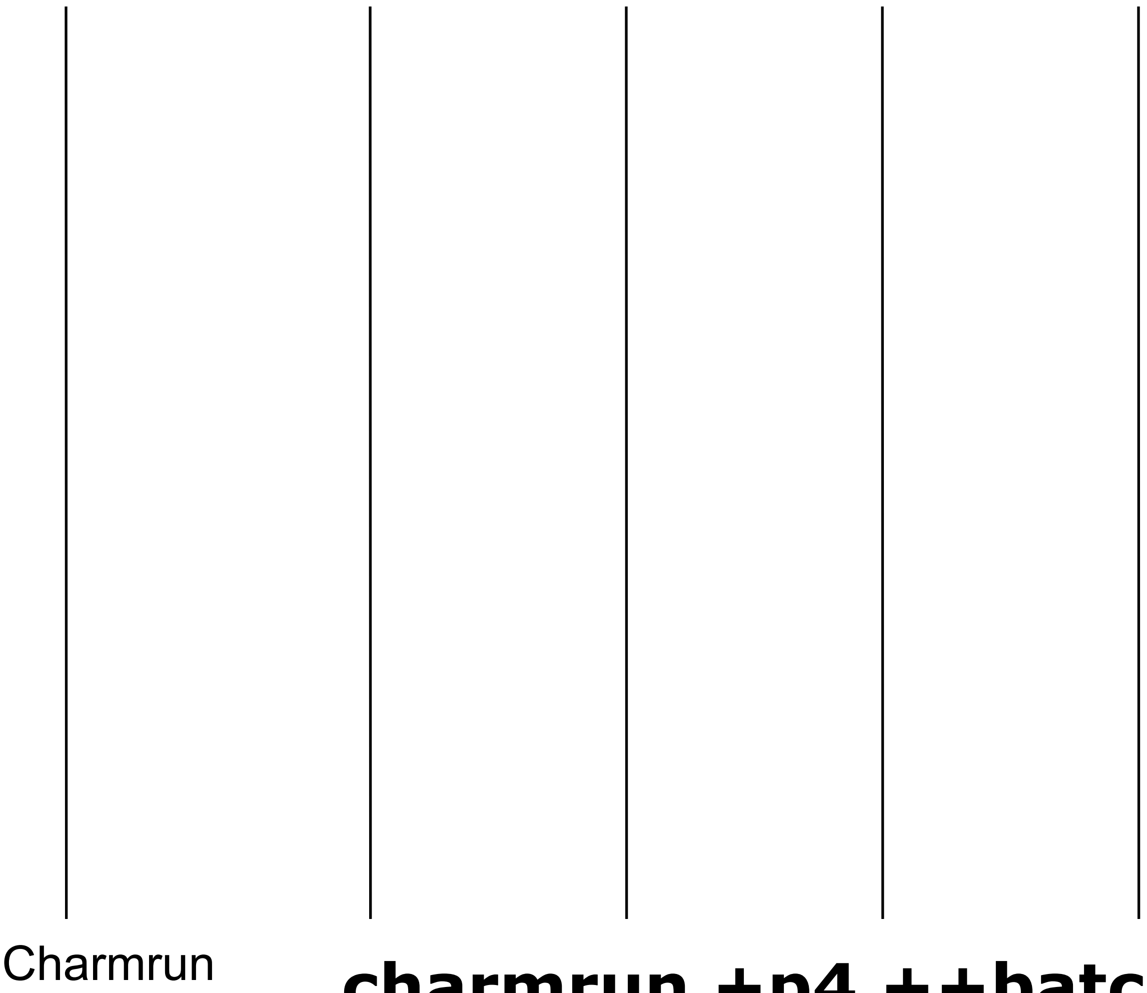
charmrun +p4 ./pgm

How Charmrun Works?



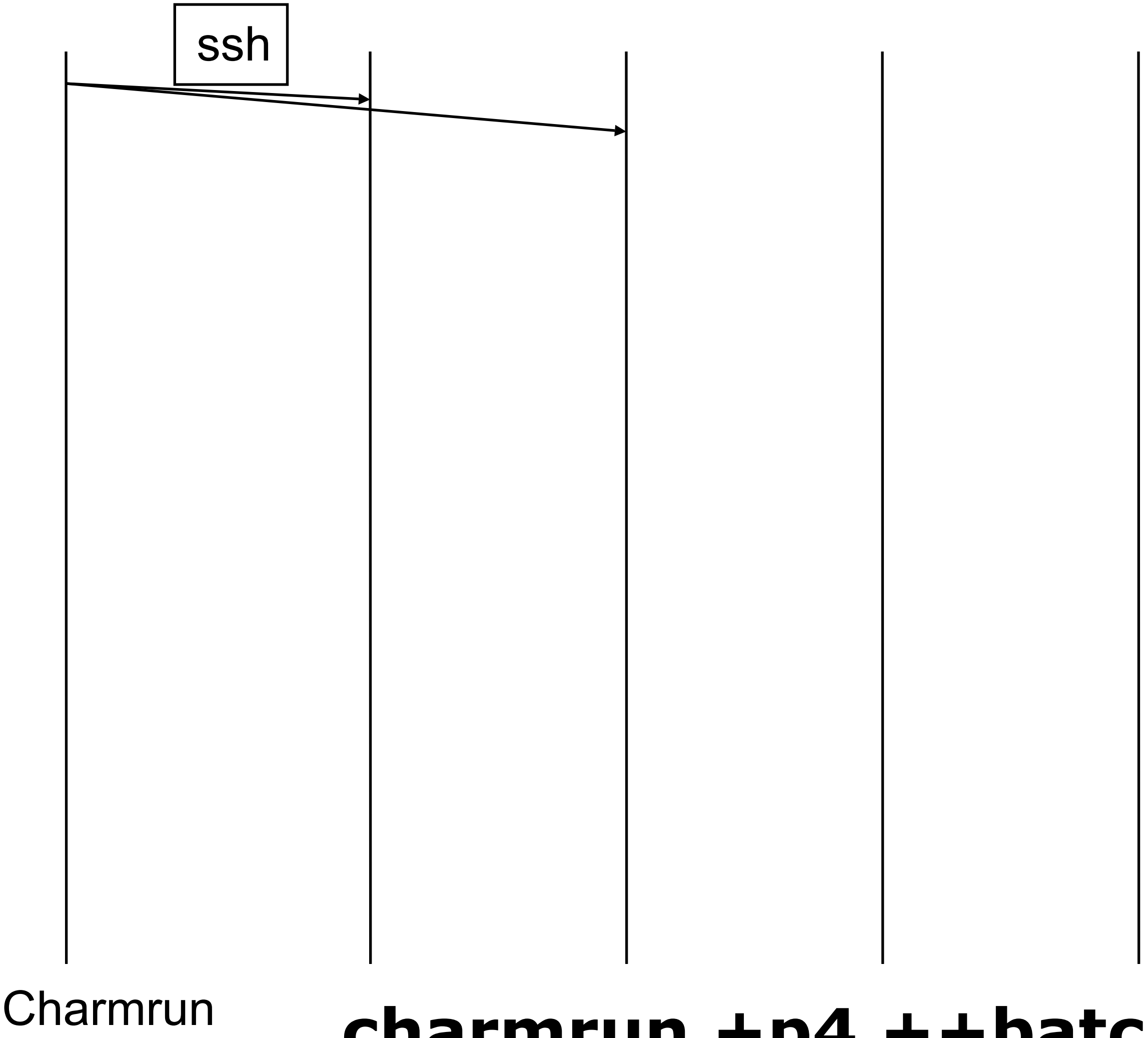
charmrun +p4 ./pgm

Charmrun (batch mode)



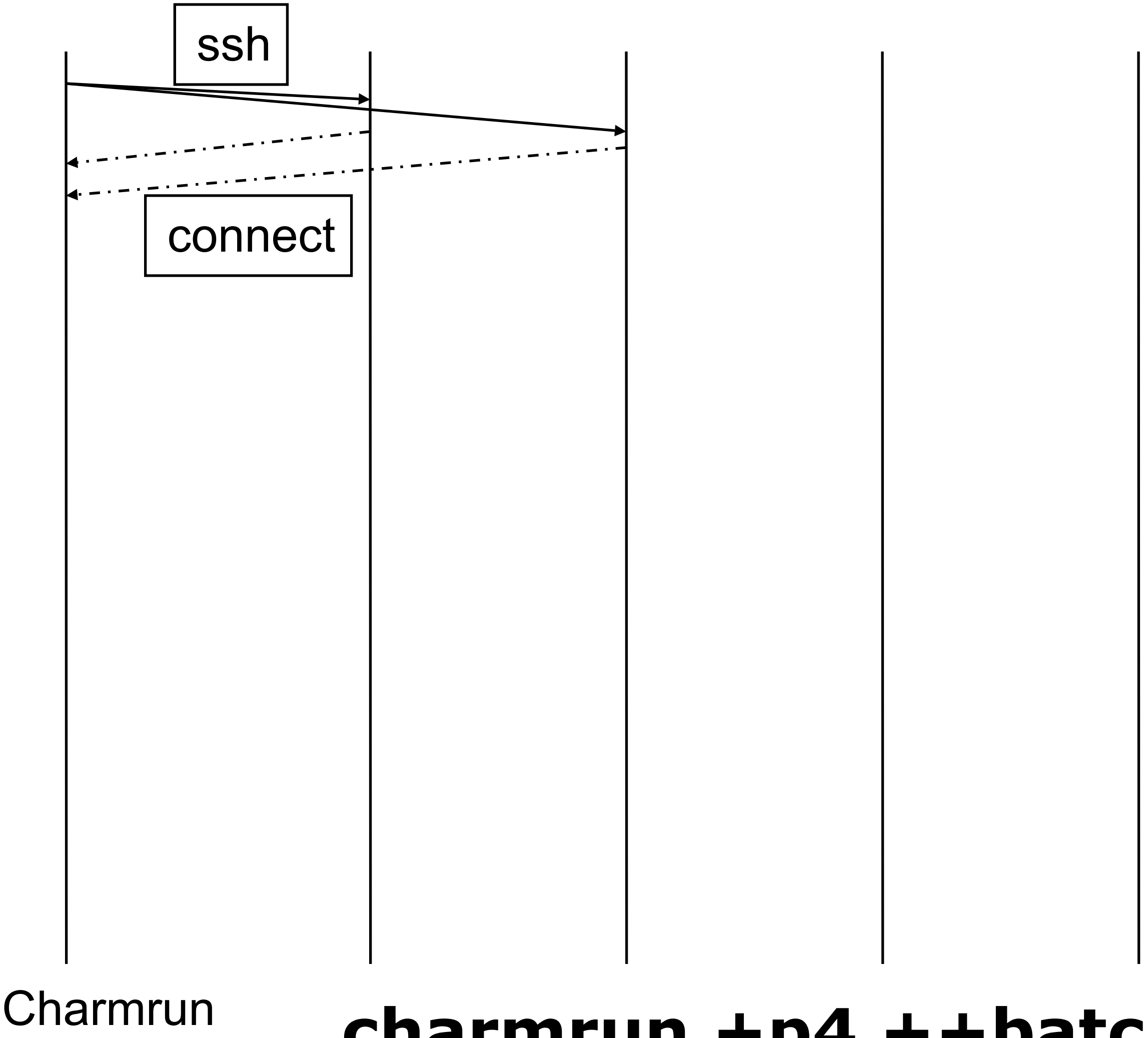
charmrun +p4 ++batch 2

Charmrun (batch mode)



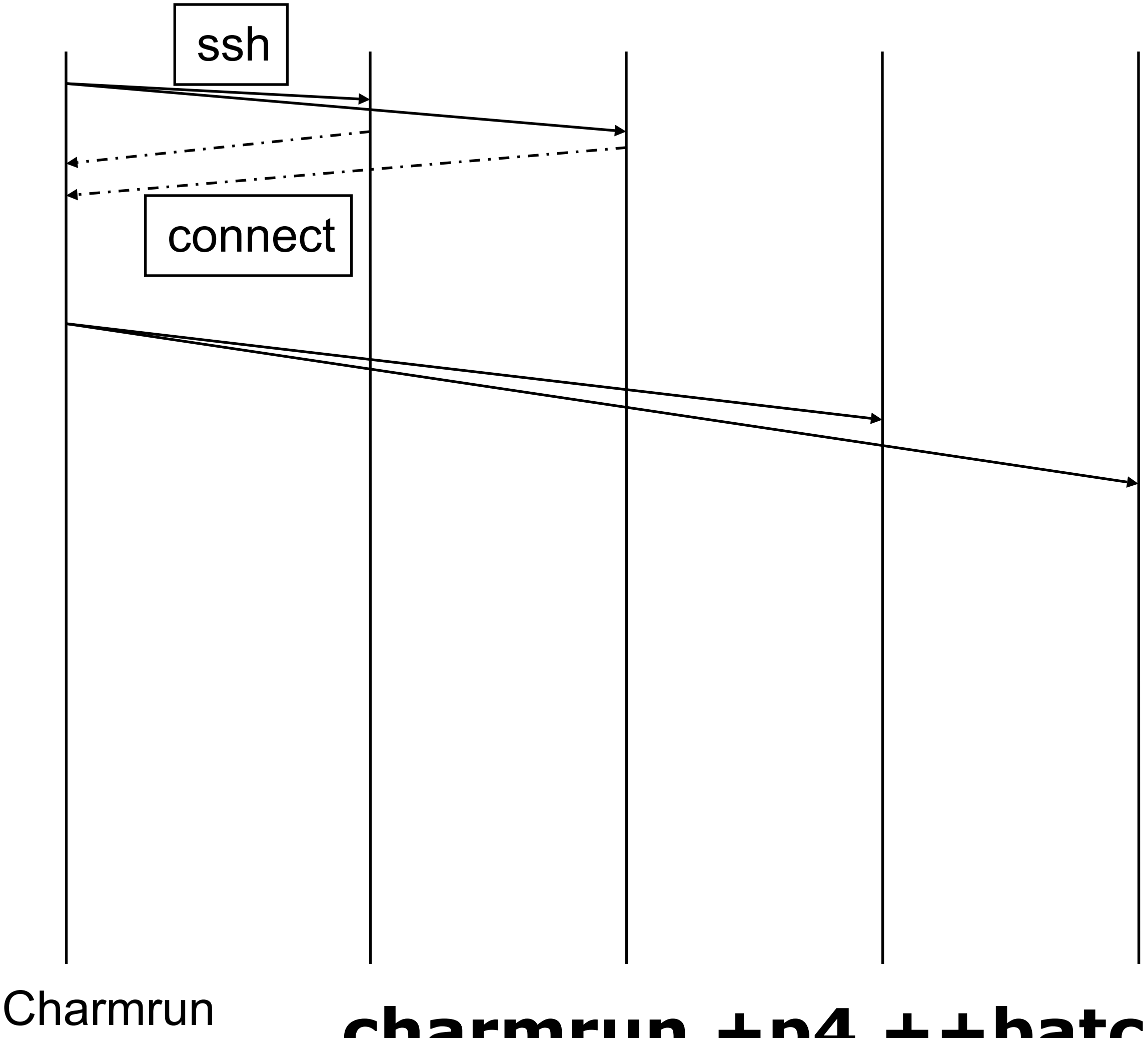
charmrun +p4 ++batch 2

Charmrun (batch mode)



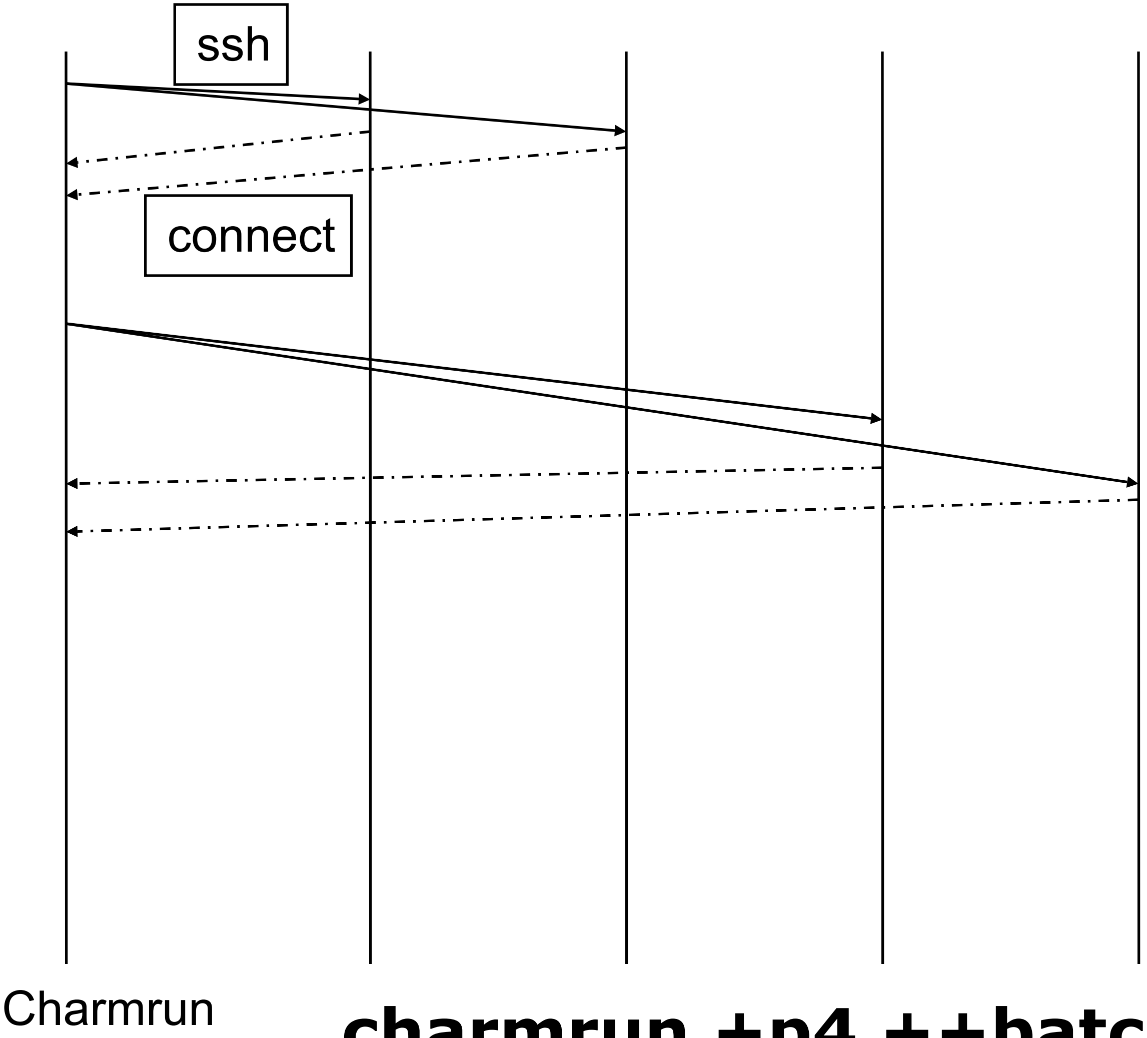
charmrun +p4 ++batch 2

Charmrun (batch mode)



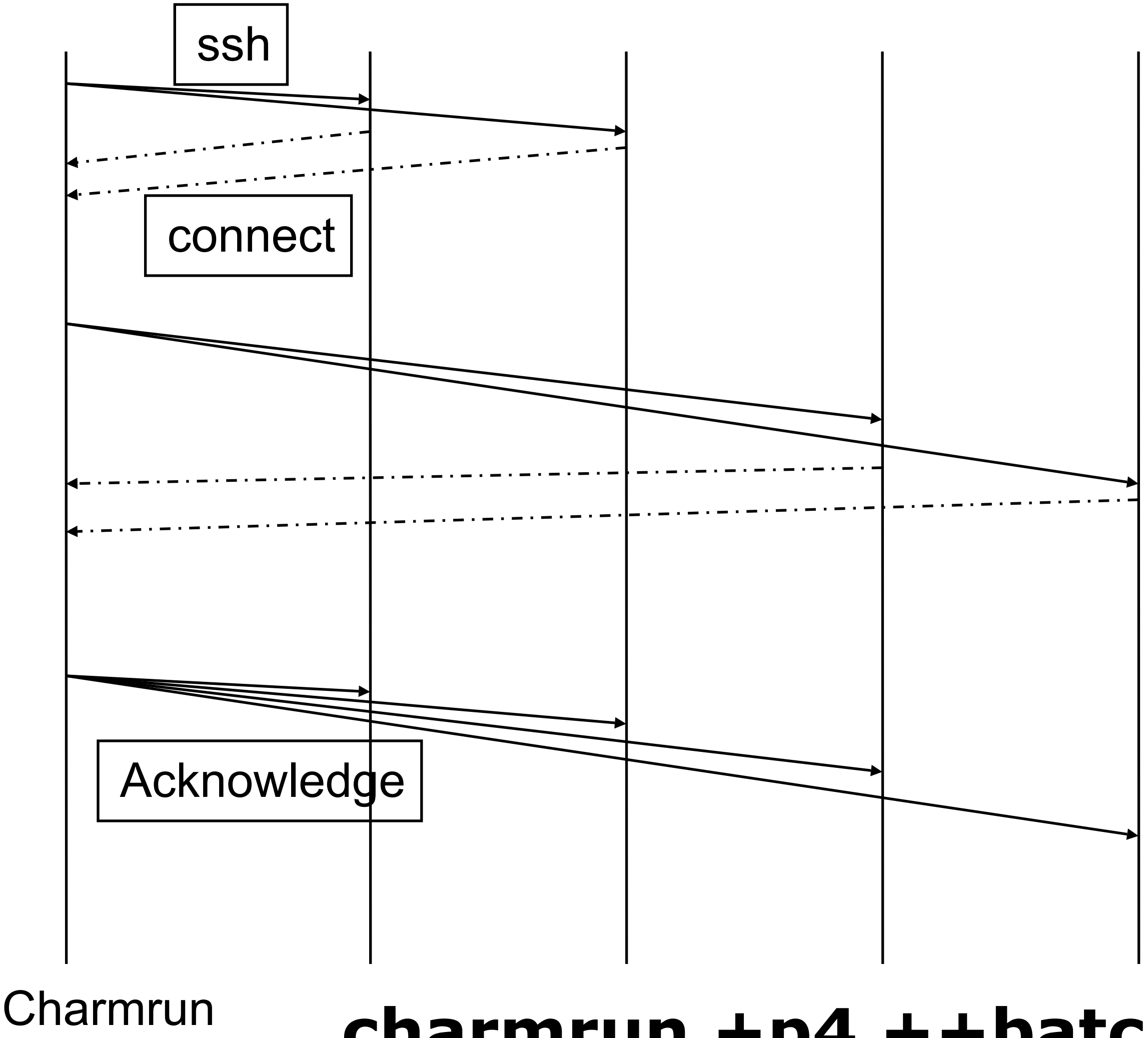
charmrun +p4 ++batch 2

Charmrun (batch mode)



charmrun +p4 ++batch 2

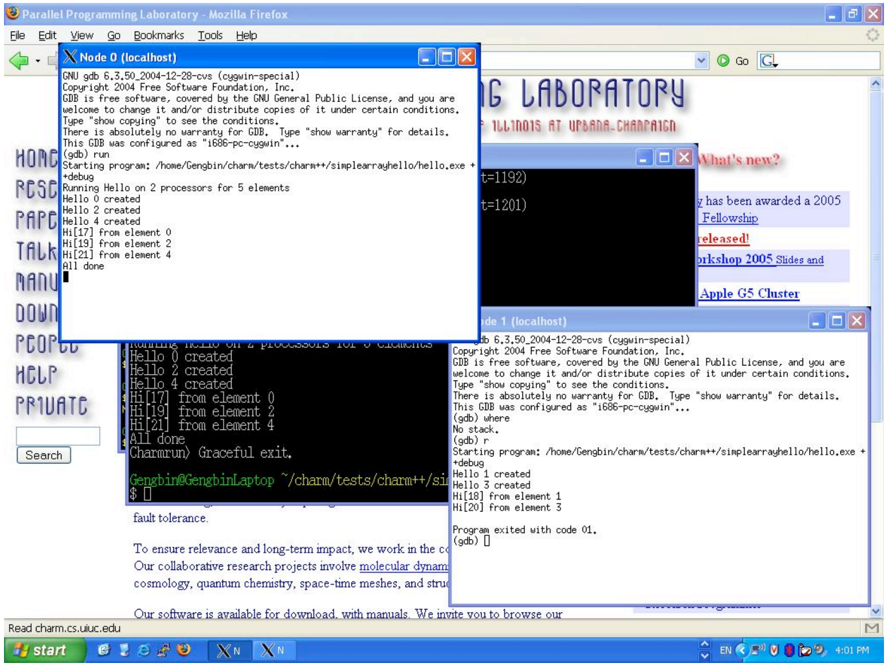
Charmrun (batch mode)



charmrun +p4 ++batch 2

Debugging Charm++ Applications

- printf
- Gdb
  - Sequentially (standalone mode)
    - gdb ./pgm +vp16
  - Attach gdb manually
  - Run debugger in xterm
    - charmrun +p4 pgm ++debug
    - charmrun +p4 pgm ++debug-no-pause
  - Memory paranoid
    - -memory paranoid
  - Parallel debugger



```
GNU gdb 6.3.50_2004-12-28-cvs (cygwin-special)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-cygwin"...
(gdb) run
Starting program: /home/Gengbin/charm/tests/charm++/simplearrayhello/hello.exe +
+debug
Running Hello on 2 processors for 5 elements
Hello 0 created
Hello 2 created
Hello 4 created
Hi[17] from element 0
Hi[19] from element 2
Hi[21] from element 4
All done
```

```
Hello 0 created
Hello 2 created
Hello 4 created
Hi[17] from element 0
Hi[19] from element 2
Hi[21] from element 4
All done
Charmrun> Graceful exit.

Gengbin@GengbinLaptop ~/charm/tests/charm++/si
$
```

```
db 6.3.50_2004-12-28-cvs (cygwin-special)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-cygwin"...
(gdb) where
No stack.
(gdb) r
Starting program: /home/Gengbin/charm/tests/charm++/simplearrayhello/hello.exe +
+debug
Hello 1 created
Hello 3 created
Hi[18] from element 1
Hi[20] from element 3

Program exited with code 01.
(gdb)
```

Advanced Messaging

PARALLEL
PROGRAMMING LAB

Prioritized Execution

- Charm++ scheduler
  - Default - FIFO (oldest message)
- Prioritized execution
  - If several messages available, Charm will process the messages in the order of their priorities
- Very useful for speculative work, ordering timestamps, etc...

Priority Classes

- Charm++ scheduler has three queues: high, default, and low
- As signed integer priorities:
  - High -MAXINT to -1
  - Default 0
  - Low 1 to +MAXINT
- As unsigned bitvector priorities:
  - 0x0000 Highest priority -- 0x7FFF
  - 0x8000 Default priority
  - 0x8001 -- 0xFFFF Lowest priority

Prioritized Messages

- Number of priority bits passed during message allocation

```
FooMsg * msg = new (size, nbits) FooMsg;
```

  - Priorities stored at the end of messages

- Signed integer priorities

```
*CkPriorityPtr(msg)=-1;
CkSetQueueing(msg, CK_QUEUEING_IFIFO);
```

- Unsigned bitvector priorities

```
CkPriorityPtr(msg)[0]=0x7fffffff;
CkSetQueueing(msg, CK_QUEUEING_BFIFO);
```

Prioritized Marshalled Messages

- **Pass "CkEntryOptions" as last parameter**
- **For signed integer priorities:**

```
CkEntryOptions opts;
opts.setPriority(-1);
fooProxy.bar(x,y,opts);
```

- **For bitvector priorities:**

```
CkEntryOptions opts;
unsigned int prio[2]={0x7FFFFFFF,0xFFFFFFFF};
opts.setPriority(64,prio);
fooProxy.bar(x,y,opts);
```

Advanced Message Features

- Read-only messages
  - Entry method agrees not to modify or delete the message
  - Avoids message copy for broadcasts, saving time
- Inline messages
  - Direct method invocation if on local processor
- Expedited messages
  - Message do not go through the charm++ scheduler (ignore any Charm++ priorities)
- Immediate messages
  - Entries are executed in an interrupt or the communication thread
  - Very fast, but tough to get right
  - Immediate messages only currently work for NodeGroups and Group (non-smp)

Read-Only, Expedited, Immediate

- **All declared in the .ci file**

```
{
   entry [nokeep] void foo_readonly(Msg *);
   entry [inline] void foo_inl(Msg *);
   entry [expedited] void foo_exp(Msg *);
   entry [immediate] void foo_imm(Msg *);
   ...
};
```

Interface File (ci)

Interface File Example

```
mainmodule hello {
  include “myType.h”

  initnode void myNodeInit();
  initproc void myInit();

  mainchare mymain {
    entry mymain(CkArgMsg *m);
  };


  array[1D] foo {
    entry foo(int problemNo);
    entry void bar1(int x);
    entry void bar2(myType x);
  };
};
```

Include and Initcall

- Include
  - Include an external header files
- Initcall
  - User plugging code to be invoked in Charm++’s startup phase
  - Initnode
    - Called once on every node
  - Initproc
    - Called once on every processor
  - Initnode calls are called before Initproc calls

Entry Attributes

- Threaded
  - Function is invoked in a CthThread
- Sync
  - Blocking methods, can return values as a message
  - Caller must be a thread
- Exclusive
  - For Node Group
  - Do not execute while other exclusive entry methods of its node group are executing in the same node
- Notrace
  - Invisible to trace projections
  - entry [notrace] void recvMsg(multicastGrpMsg *m);

Groups/Node Groups

Groups and Node Groups

- Groups
  - Similar to arrays:
    - Broadcasts, reductions, indexing
  - But not completely like arrays:
    - Non-migratable; one per processor
  - Exactly one representative on each processor
    - Ideally suited for system libraries
  - Historically called branch office chares (BOC)
- Node Groups
  - One per SMP node

Declarations

- .ci file

```
 group mygroup {
      entry mygroup(); //Constructor
      entry void foo(foomsg *); //Entry method
 };
 nodegroup mynodegroup {
      entry mynodegroup(); //Constructor
      entry void foo(foomsg *); //Entry method
 };
```

- C++ file

```
class mygroup : public Group {
                  mygroup() {}
      void foo(foomsg *m) { CkPrintf("Do Nothing");}
};
class mynodegroup : public NodeGroup {
                  mynodegroup() {}
      void foo(foomsg *m) { CkPrintf("Do Nothing");}
};
```

Creating and Calling Groups

- Creation

```
p = CProxy_mygroup::ckNew();
```

- Remote invocation

```
p.foo(msg);    //broadcast
p[1].foo(msg);  //asynchronous
p.foo(msg, npes, pes); // list send
```

- Direct local access

```
mygroup *g=p.ckLocalBranch();
g->foo(….); //local invocation
```

  - Danger: if you migrate, the group stays behind!

Advanced Load-balancers

PARALLEL
PROGRAMMING LAB

Advanced load balancing: Writing a new strategy

- Inherit from CentralLB and implement the work(…) function

```
class foolb : public CentralLB {
    public:
        .. .. ..
        void work (CentralLB::LDStats*   stats, int count);
        .. .. ..
  };
```

LB Database

```
struct LDStats {
   ProcStats *procs;
   LDObjData* objData;
   LDCommData* commData;
   int *to_proc;
   //.. .. ..
}
//Dummy Work function which assigns all objects to
//processor 0
//Don't implement it!
void fooLB::work(CentralLB::LDStats* stats,int count){
     for(int count=0;count < nobjs; count++)
        stats.to_proc[count] = 0;
}
```

Compiling and Integration

- Edit and run Makefile_lb.sh
  - Creates Make.lb which is included by the main Makefile
- Run make depends to correct dependencies
- Rebuild charm++ and is now available in –balancer fooLB

Threads in Charm++

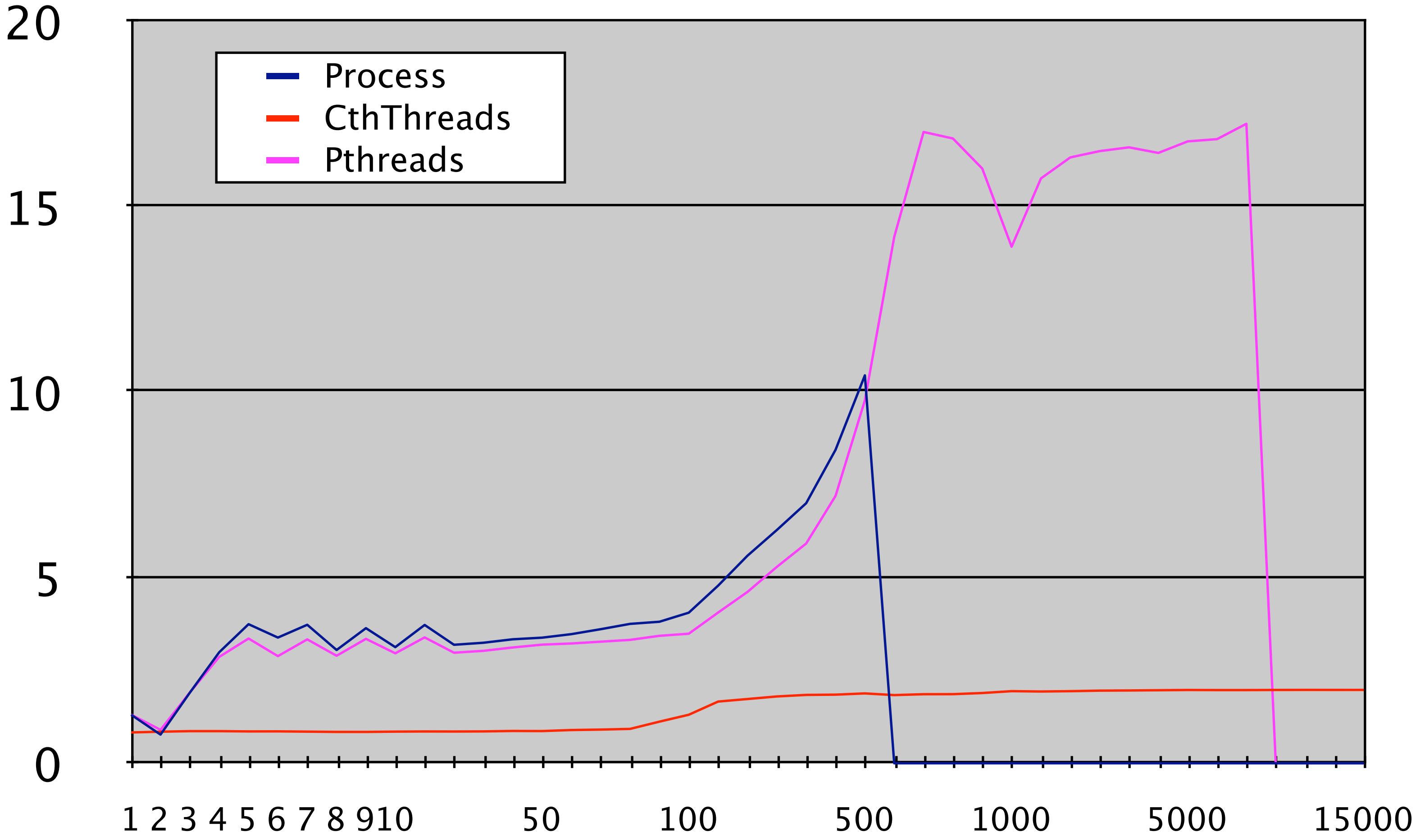
Why use Threads?

- They provide one key feature: blocking
  - Suspend execution (e.g., at message receive)
  - Do something else
  - Resume later (e.g., after message arrives)
- Example: MPI_Recv, MPI_Wait semantics
- Function call interface more convenient than message-passing
  - Regular call/return structure (no CkCallbacks) with complete control flow
  - Allows blocking in middle of deeply

Why not use Threads?

- Slower
  - Around 1us context-switching overhead unavoidable
  - Creation/deletion perhaps 10us
- Migration more difficult
  - State of thread is scattered through stack, which is maintained by compiler
  - By contrast, state of object is maintained by users
- Thread disadvantages form the motivation to use SDAG (later)

Context Switch Cost



What are (Converse) Threads?

- One flow of control (instruction stream)
  - Machine Registers & program counter
  - Execution stack
- Like pthreads (kernel threads)
- Only different:
  - Implemented at user level (in Converse)
  - Scheduled at user level; non-preemptive
  - Migratable between nodes

How do I use Threads?

- Many options:
  - AMPI
    - Always uses threads via TCharm library
  - Charm++
    - [threaded] entry methods run in a thread
    - [sync] methods
  - Converse
    - C routines CthCreate/CthSuspend/CthAwaken
    - Everything else is built on these
    - Implemented using
      - SYSV makecontext/setcontext
      - POSIX setjmp/alloca/longjmp
      - Assembly code

How do I use Threads (example)

- **Blocking API routine: find array element**

```
int requestFoo(int src) {
  myObject *obj=...;
  return obj->fooRequest(src)
}
```

- **Send request and suspend**

```
int myObject::fooRequest(int src) {
  proxy[dest].fooNetworkRequest(thisIndex);
  stashed_thread=CthSelf();
  CthSuspend();   // -- blocks until awaken call --
  return stashed_return;
}
```

- **Awaken thread when data arrives**

```
void myObject::fooNetworkResponse(int ret) {
  stashed_return=ret;
  CthAwaken(stashed_thread);
}
```

How do I use Threads (example)

- **Send request, suspend, recv, awaken, return**

```
int myObject::fooRequest(int src) {
  proxy[dest].fooNetworkRequest(thisIndex);
  stashed_thread=CthSelf();
  CthSuspend();
         void myObject::fooNetworkResponse(int ret) {
           stashed_return=ret;
           CthAwaken(stashed_thread);
         }

  return stashed_return;
}
```

Thread Migration

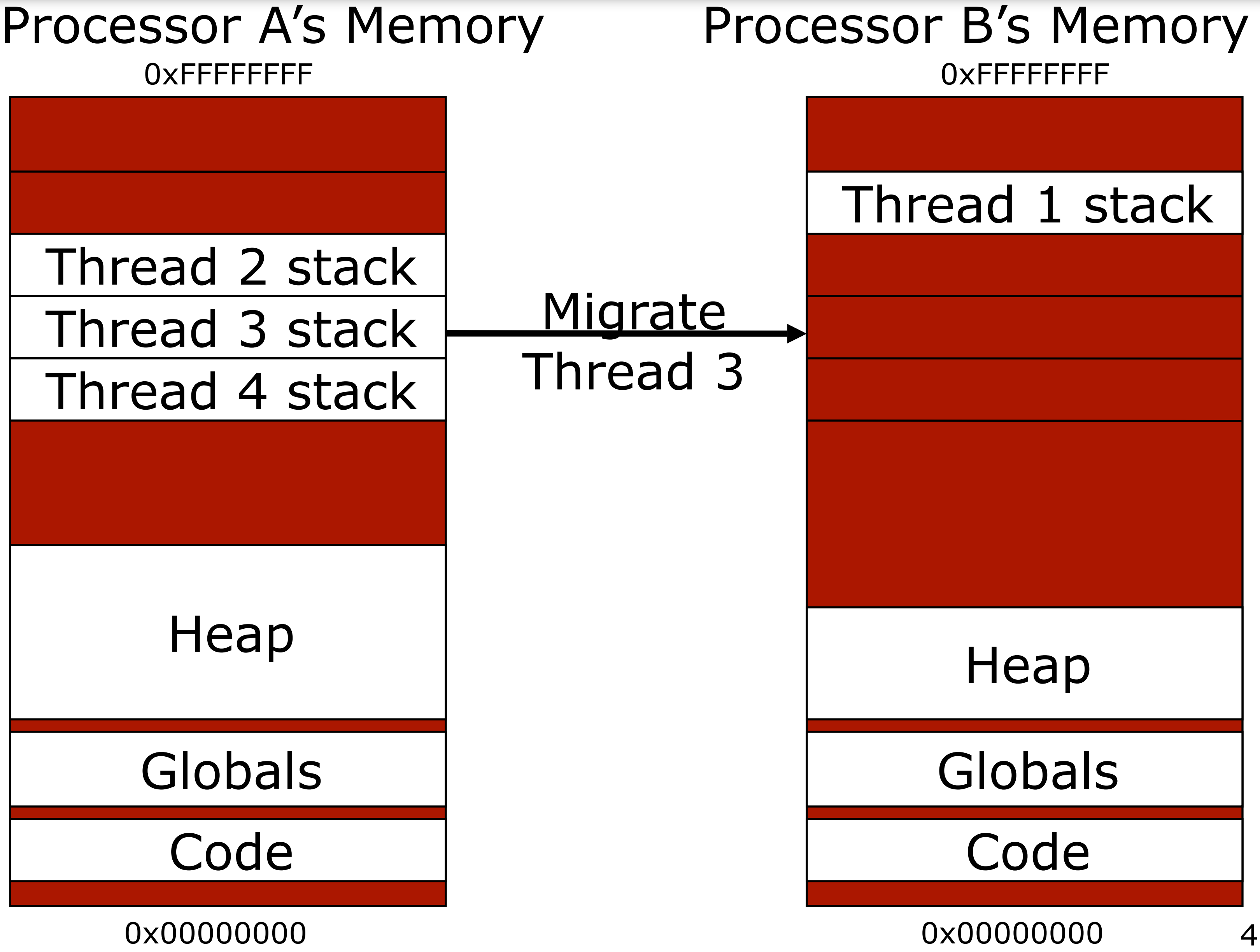
Stack Data

- The stack is used by the compiler to track function calls and provide temporary storage
  - Local Variables
  - Subroutine Parameters
  - C "alloca" storage
- Most of the variables in a typical application are stack data
- Stack is allocated by Charm run-time as heap memory (+stacksize)

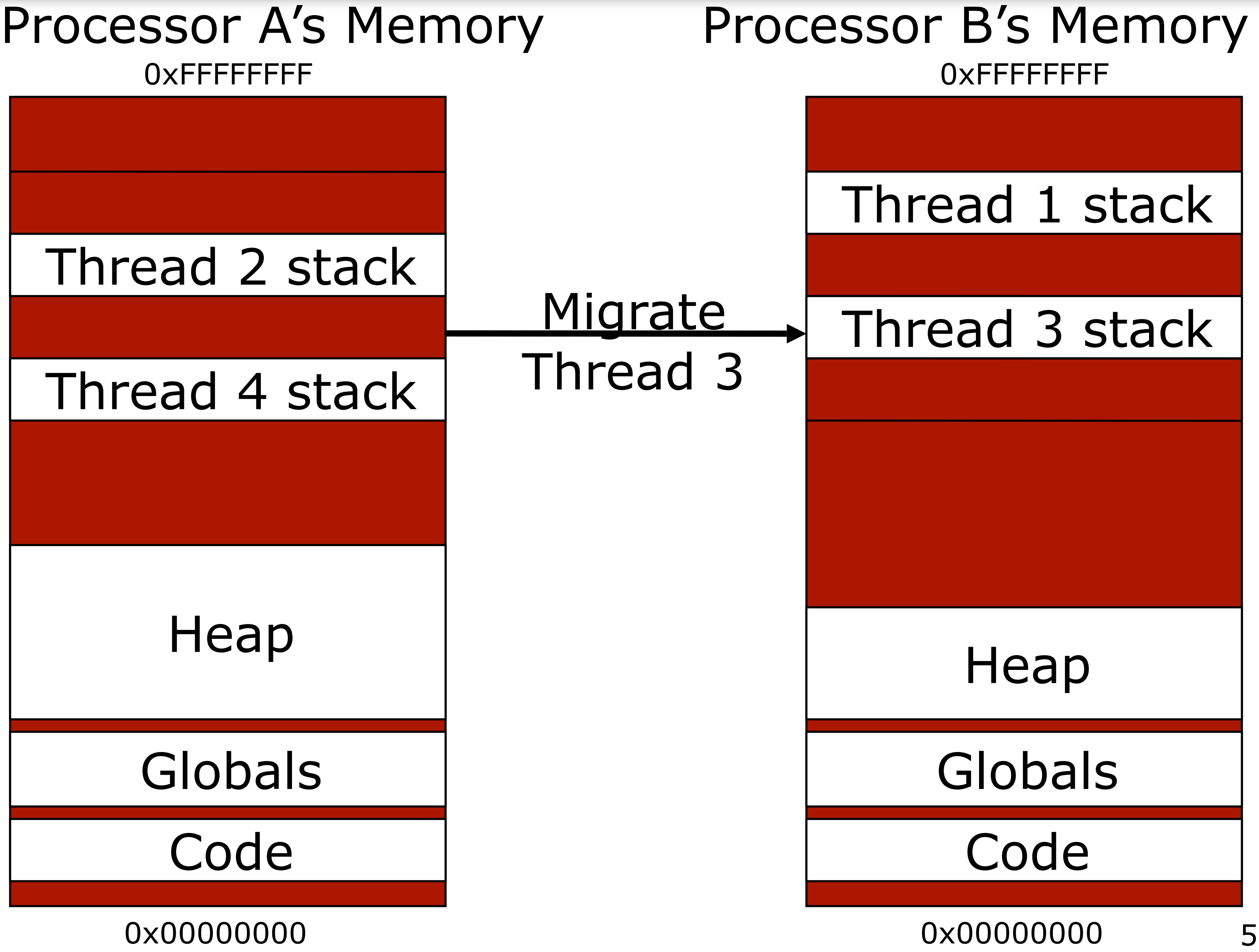
Migrate Stack Data

- Without compiler support, cannot change stack’s address
  - Because we can’t change stack’s interior pointers (return frame pointer, function arguments, etc.)
- Existing pointers to addresses in original stack become invalid
- Solution: “isomalloc” addresses
  - Reserve address space on every processor for every thread stack
  - Use *mmap* to scatter stacks in virtual memory efficiently
  - Idea comes from PM^2

Migrate Stack Data



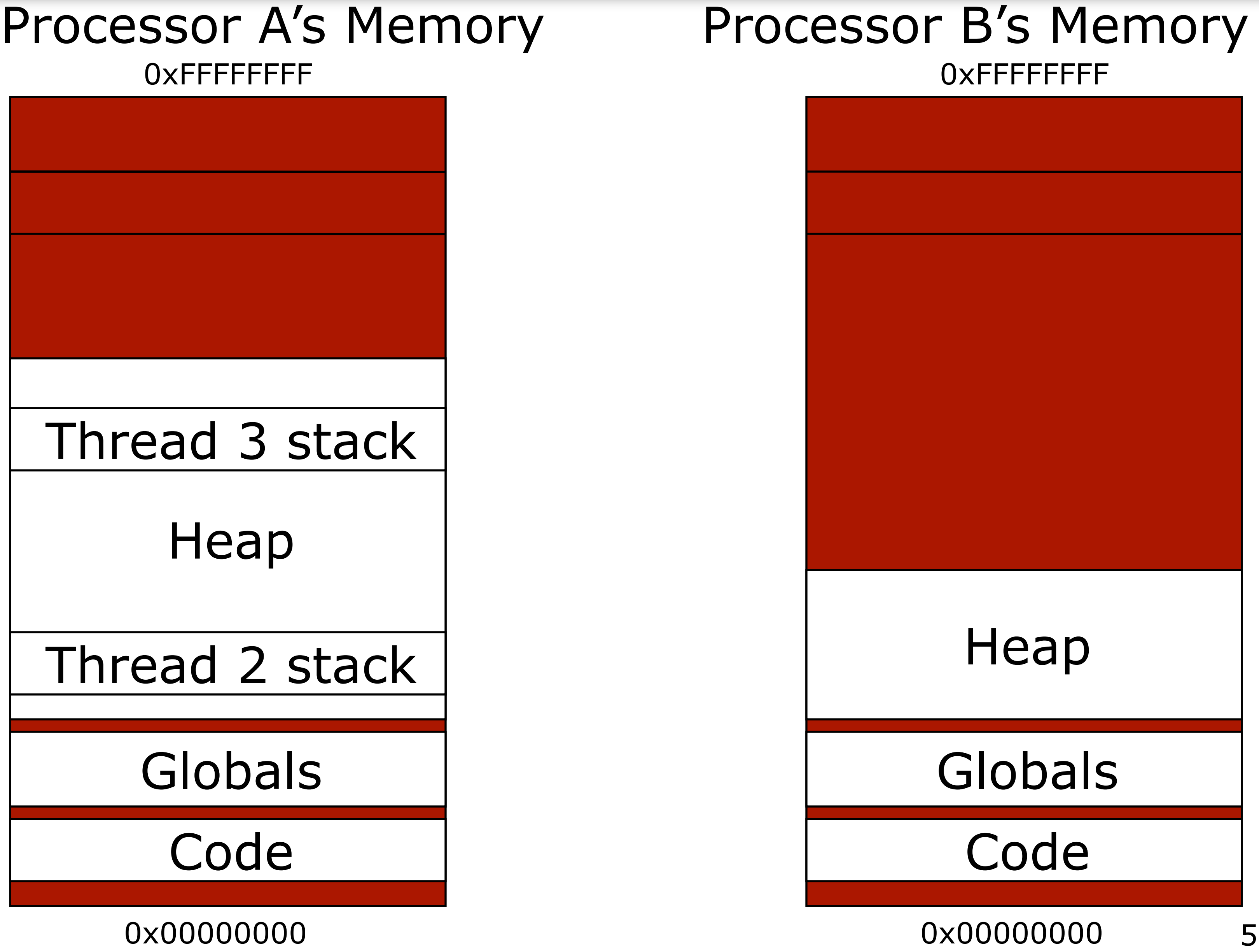
Migrate Stack Data: Isomalloc



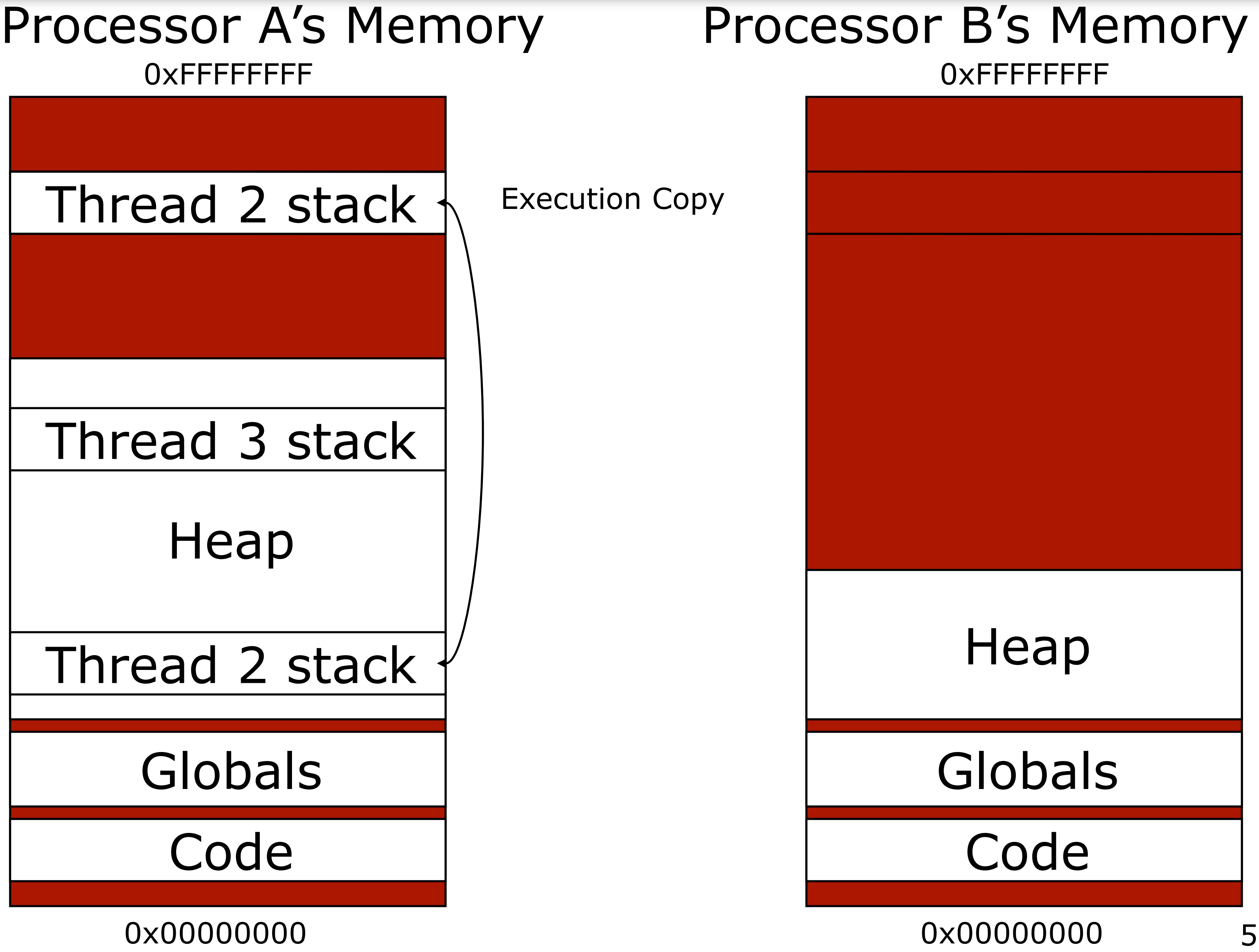
Migrate Stack Data

- Isomalloc is a completely automatic solution
  - No changes needed in application or compilers
  - Just like a software shared-memory system, but with proactive paging
- But has a few limitations
  - Depends on having large quantities of virtual address space (best on 64-bit)
    - 32-bit machines can only have a few gigs of isomalloc stacks across the whole machine
  - Depends on unportable *mmap*
    - Which addresses are safe? (We must guess!)
    - What about Windows? Or Blue Gene?

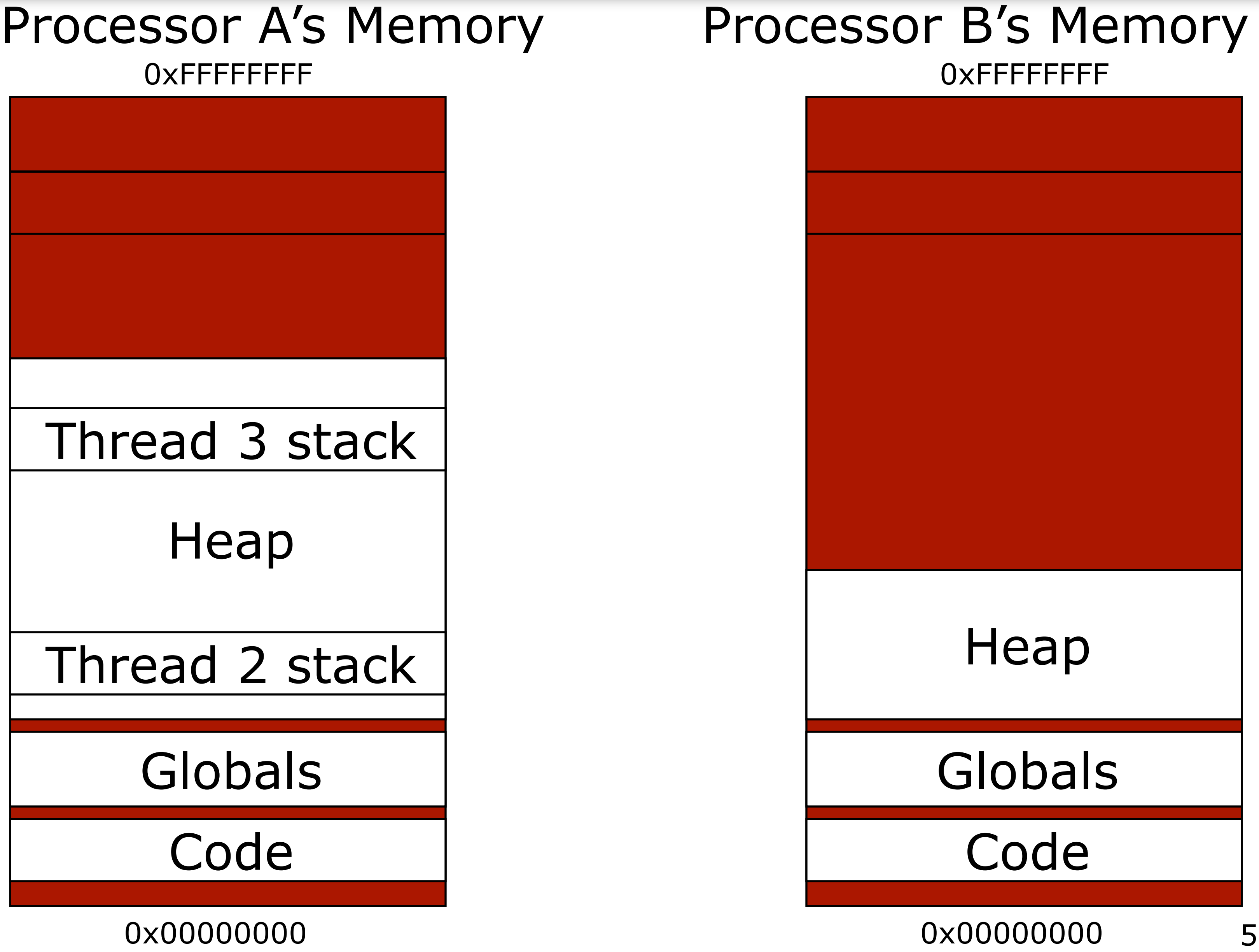
Aliasing Stack Data



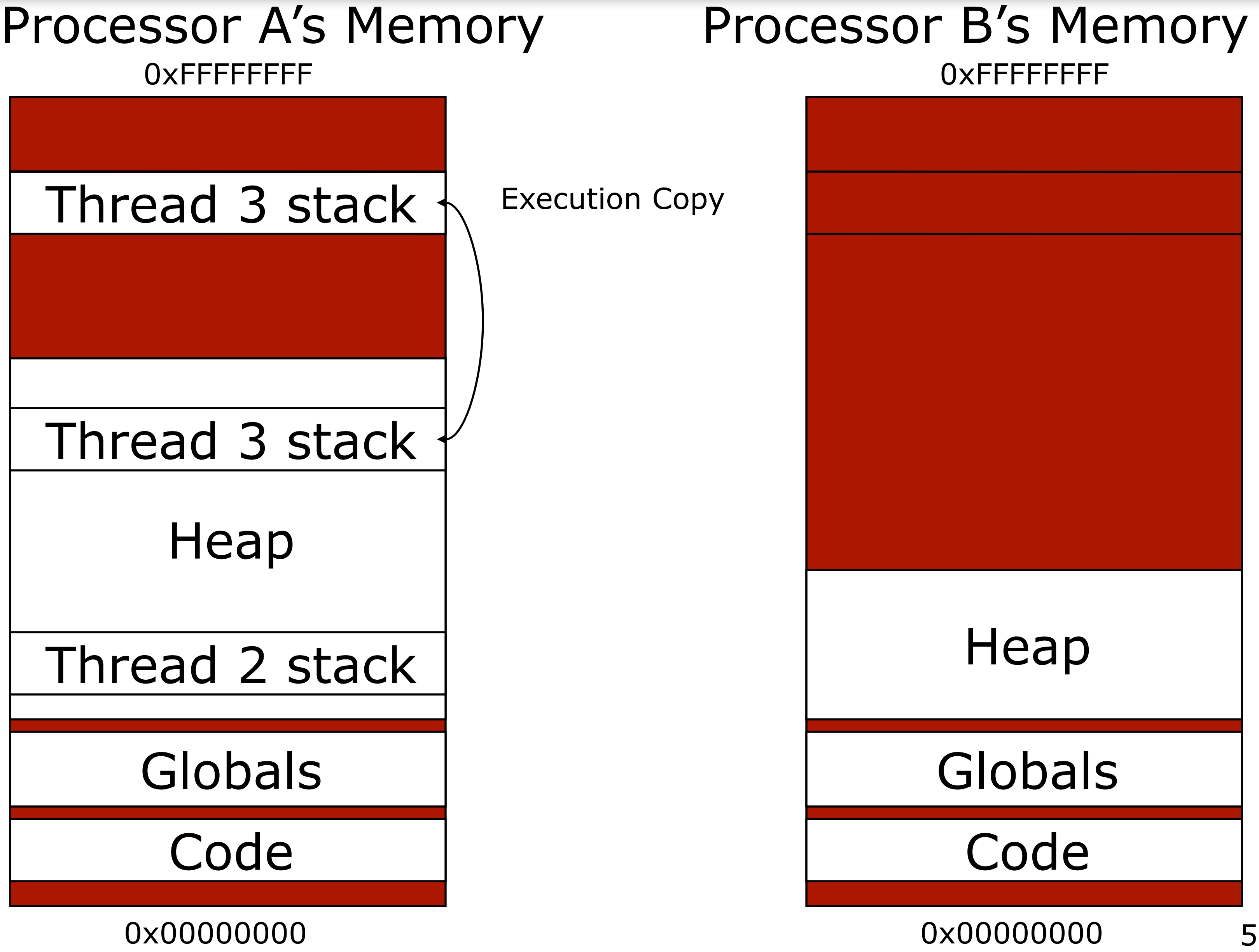
Aliasing Stack Data: Run Thread 2



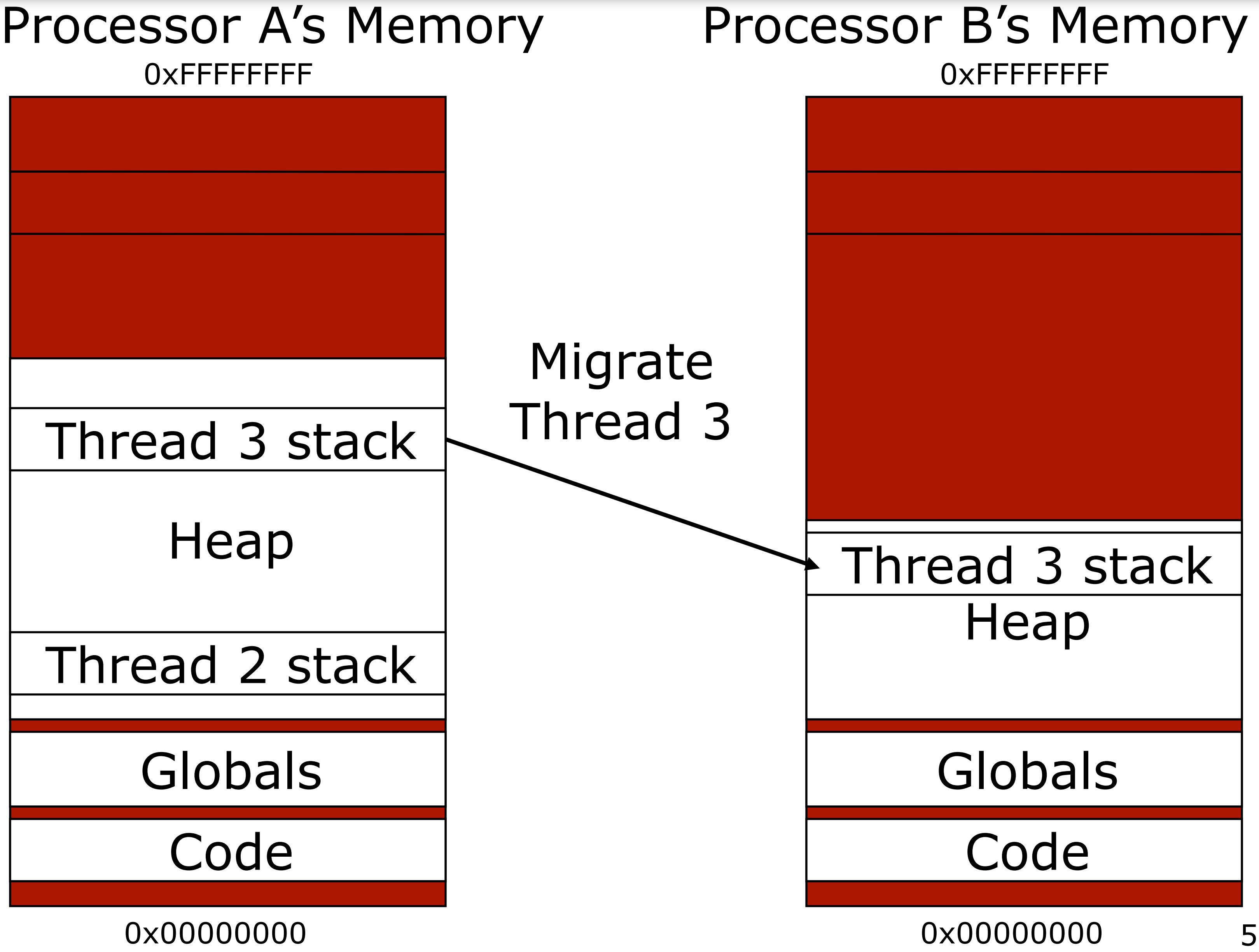
Aliasing Stack Data



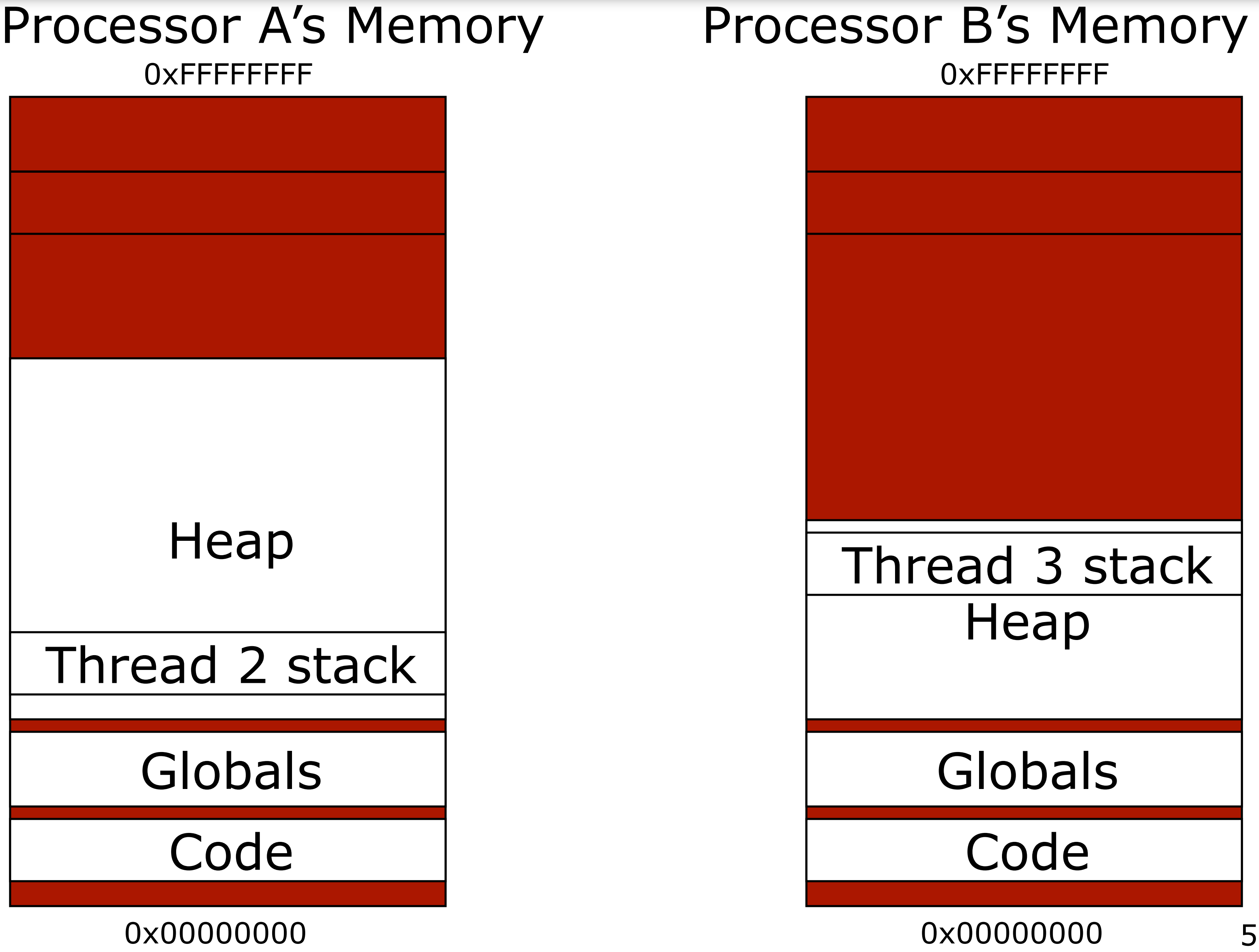
Aliasing Stack Data: Run Thread 3



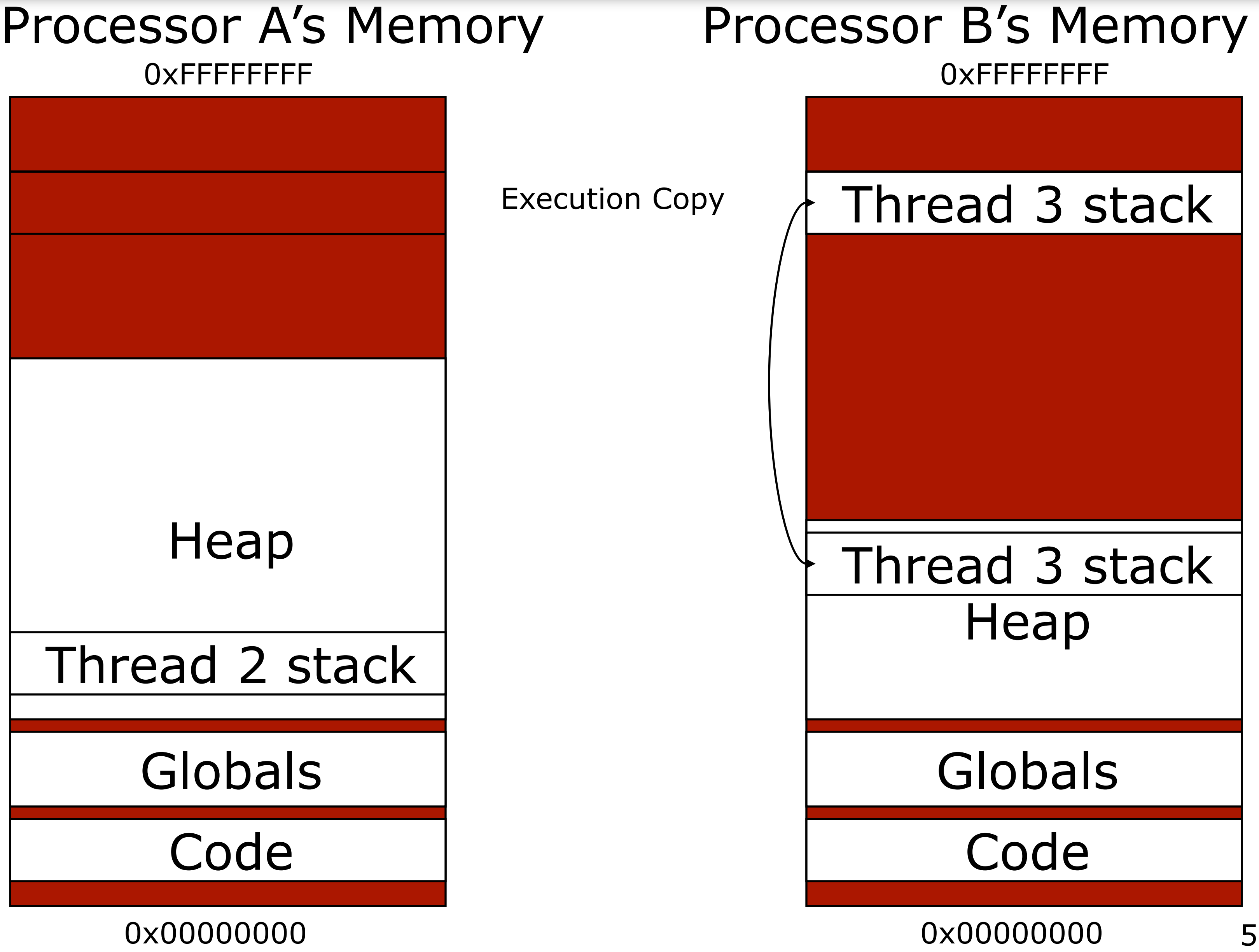
Aliasing Stack Data



Aliasing Stack Data



Aliasing Stack Data



Aliasing Stack Data

- Does not depend on having large quantities of virtual address space
  - Works well on 32-bit machines
- Requires only one mmap’d region at a time
  - Works even on Blue Gene!
- Downsides:
  - Thread context switch requires munmap/ mmap (3us)
  - Can only have one thread running at a time (so no SMP’s!)
- “-thread memoryalias” link time option

Heap Data

- Heap data is any dynamically allocated data
  - C “malloc” and “free”
  - C++ “new” and “delete”
  - F90 “ALLOCATE” and “DEALLOCATE”
- Arrays and linked data structures are almost always heap data

Migrate Heap Data

- Automatic solution: isomalloc all heap data just like stacks!
  - "-memory isomalloc" link option
  - Overrides *malloc/free*
  - No new application code needed
  - Same limitations as isomalloc; page allocation granularity (huge!)
- Manual solution: application moves its heap data
  - Need to be able to <u>size</u> message buffer, <u>pack</u> data into message, and <u>unpack</u> on other side
  - "pup" abstraction does all three

Delegation

Delegation

- Customized implementation of messaging
  - Enables Charm++ proxy messages to be forwarded to a delegation manager group
- Delegation manager
  - trap calls to proxy sends and apply optimizations
- Delegation manager must inherit from CkDelegateMgr class
- User program must to call
  - proxy.ckDelegate(mgrID);

Delegation Interface

- .ci file

```
group MyDelegateMgr {
        entry MyDelegateMgr();  //Constructor
 };
```

- .h file

```
class MyDelegateMgr : public CkDelegateMgr {
  MyDelegateMgr();
  void ArraySend(...,int ep,void *m,const
  CkArrayIndexMax &idx,CkArrayID a);
  void ArrayBroadcast(..);
  void ArraySectionSend(.., CkSectionID &s);
      ................ . .
      ................ . .
}
```

Array Multicast

Array Multicast/reduction library

- **Array section – a subset of chare array**
- **Array section creation**
  - **Enumerate array indices**

```
CkVec<CkArrayIndex3D> elems;    // add array indices
  for (int i=0; i<10; i++)
    for (int j=0; j<20; j+=2)
      for (int k=0; k<30; k+=2)
         elems.push_back(CkArrayIndex3D(i, j, k));
  CProxySection_Hello proxy = CProxySection_Hello::ckNew(helloArrayID, elems.getVec(), elems.size());
```

  - **Alternatively, one can do the same thing by providing (lbound:ubound:stride) for each dimension:**

```
CProxySection_Hello proxy = CProxySection_Hello::ckNew(helloArrayID, 0, 9, 1, 0, 19, 2, 0, 29, 2);
```

    - **The above code creates a section proxy that contains array elements of [0:9, 0:19:2, 0:29:2].**

  - **For user-defined array index other than CkArrayIndex1D to CkArrayIndex6D, one needs to use the generic array index type: CkArrayIndexMax.**

```
CkArrayIndexMax *elems;    // add array indices
int numElems;
CProxySection_Hello proxy = CProxySection_Hello::ckNew(helloArrayID, elems, numElems);
```

Array Section Multicast

- **Once have the array section proxy**
  - **do multicast to all the section members:**
    - CProxySection_Hello proxy;
      proxy.foo(msg)   // multicast
  - **send messages to one member using its local index**
    - proxy[0].foo(msg)

Array Section Multicast

- **Multicast via delegation**
  - **CkMulticast communication library**
  - CProxySection_Hello sectProxy = CProxySection_Hello::ckNew();
    CkGroupID mCastGrpId = CProxy_CkMulticastMgr::ckNew();
    CkMulticastMgr *mcastGrp = CProxy_CkMulticastMgr
    (mCastGrpId).ckLocalBranch();

```
sectProxy.ckSectionDelegate(mCastGrpId);  // initialize proxy

sectProxy.foo(...);          //multicast via delegation
```

- **Note, to use CkMulticast library, all multicast messages must inherit from CkMcastBaseMsg, as following:**

```
class HiMsg : public CkMcastBaseMsg, public CMessage_HiMsg
{
    public:
    int *data;
};
```

Array Section Reduction

- **Section reduction with delegation**
- **Use default reduction callback**

```
CProxySection_Hello sectProxy;
CkMulticastMgr *mcastGrp = CProxy_CkMulticastMgr
    (mCastGrpId).ckLocalBranch();

mcastGrp->setReductionClient(sectProxy, new CkCallback(...));
```

- **Reduction**

```
CkGetSectionInfo(sid, msg);
CkCallback cb(CkIndex_myArray::foo(NULL),thisProxy);
mcastGrp->contribute(sizeof(int), &data, CkReduction::sum_int, sid,
    cb);
```

With Migration

- Works with migration
  - When intermediate nodes migrate
    - When migrates, multicast tree will be automatically rebuilt
  - Root processor
    - Application needs to initiate the rebuild
    - Will change to automatic in future

SDAG

PARALLEL PROGRAMMING LAB

Structured Dagger

- What is it?
  - A coordination language built on top of Charm++
  - Express control flow in interface file
- Motivation
  - Charm++'s asynchrony is efficient and reliable, but tough to program
    - Split phase - Flags, buffering, out-of-order receives, etc.
  - Threads are easy to program, but less efficient and less reliable
    - Implementation complexity
    - Porting headaches
  - Want benefits of both!

Structured Dagger Constructs

- when <method list> {code}
  - Do not continue until method is called
    - Internally generates flags, checks, etc.
    - Does not use threads
- atomic {code}
  - Call ordinary sequential C++ code
- if/else/for/while
  - C-like control flow
- overlap {code1 code2 ...}
  - Execute code segments in parallel
- forall
  - "Parallel Do"
  - Like a parameterized overlap

Stencil Example Using SDAG

```
array[1D] myArray {
…
entry void GetMessages () {
    when rightmsgEntry(), leftmsgEntry() {
         atomic {
           CkPrintf(“Got both left and right messages \n”);
           doWork(right, left); }
     }
};

entry void rightmsgEntry();
entry void leftmsgEntry();
…
};
```

Overlap for LeanMD Initialization

```
array[1D] myArray {
…
 entry void waitForInit(void) {
     overlap {
        when recvNumCellPairs(myMsg* pMsg) {
          atomic { setNumCellPairs(pMsg->intVal);  delete 
pMsg; }
        }
        when recvNumCells(myMsg * cMsg) {
          atomic { setNumCells(cMsg->intVal);  delete cMsg; }
        }
     }
 }
};
```

For for LeanMD timeloop

```
entry void doTimeloop(void) {
   for (timeStep_=1; timeStep_<=SimParam.NumSteps; timeStep++) {
        atomic {sendAtomPos(); }

        overlap {
          for (forceCount_=0; forceCount_<numForceMsg_; forceCount_++) {
            when recvForces(ForcesMsg* msg) { atomic {procForces(msg); } }
          }

          for (pmeCount_=0; pmeCount_<nPME; pmeCount_++) {
            when recvPME(PMEGridMsg* m) {atomic {procPME(m);}}
          }
        }

        atomic { doIntegration(); }

        if (timeForMigrate()) { ... }
   }
}
```

Thank You!

Free source, binaries, manuals, and more information at:
http://charm.cs.uiuc.edu/

Parallel Programming Lab at University of Illinois

PARALLEL
PROGRAMMING LAB