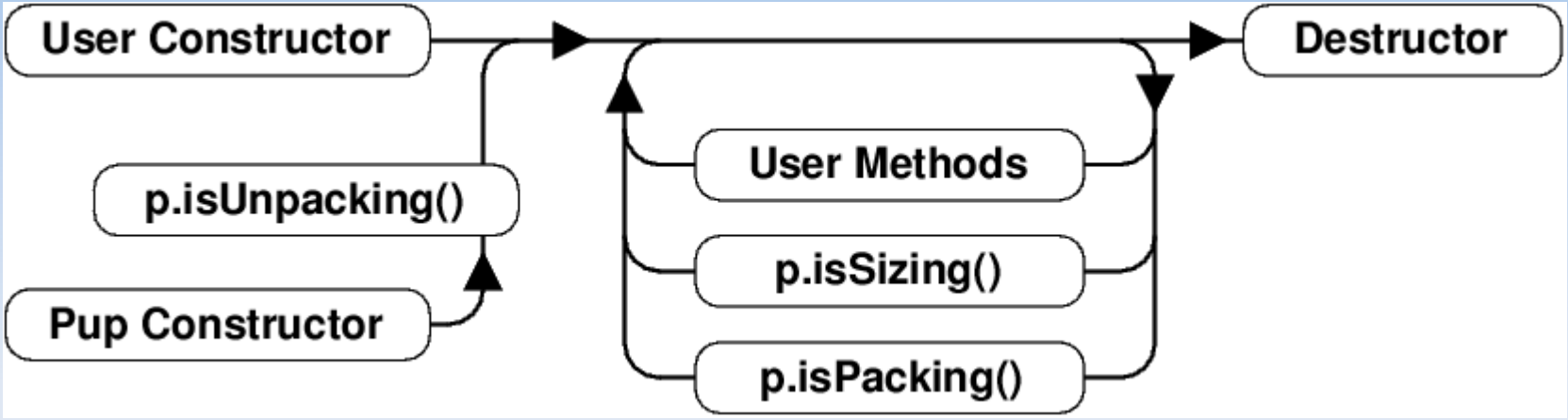
Chare Migration: motivations

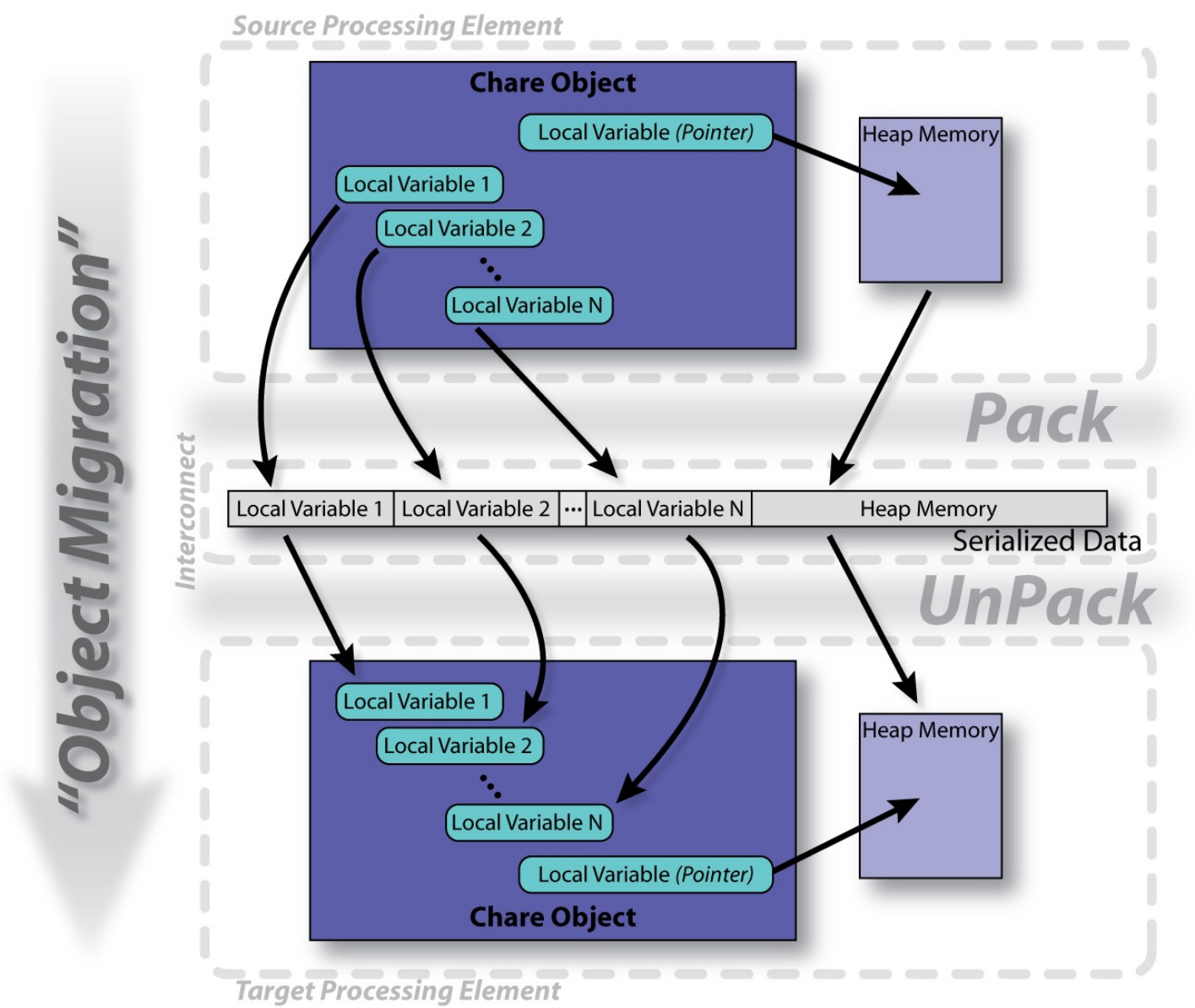
- Chares are initially placed according to a *placement map*
  - The user can specify this map
- While running, some processors might be overloaded
  - Need to rebalance the load
- Automatic checkpoint
  - Migration to disk

The life of a chare



- Migration out:
  - ckAboutToMigrate()
  - Sizing
  - Packing
  - Destructor
- Migration in:
  - Migration constructor
  - UnPacking
  - ckJustMigrated()

The PUP process



Writing a PUP routine

or:
void operator|(PUP::er &p, MyChare &c)

```
class MyChare : public CBase_MyChare {
  int a;
  float b;
  char c;
  float localArray[LOCAL_SIZE];
  int heapArraySize;
  float* heapArray;
  MyClass *pointer;

 public:
  MyChare();
  MyChare(CkMigrateMessage *msg) {};
  ~MyChare() {
    if (heapArray != NULL) {
      delete [] heapArray;
      heapArray = NULL;
    }
  }
```

```
void pup(PUP::er &p) {
  CBase_MyChare::pup(p);

  p | a;
  p | b;
  p | c;

  p(localArray, LOCAL_SIZE);

  p | heapArraySize;
  if (p.isUnpacking()) {
    heapArray = new float[heapArraySize];
  }
  p(heapArray, heapArraySize);

  int isNull = (pointer==NULL) ? 1 : 0;
  p | isNull;
  if (!isNull) {
    if (p.isUnpacking()) pointer = new MyClass();
    p | *pointer;
  }
 }
};
```

PUP: what to look for

- If variables are added to an object, update the PUP routine
- If the object allocates data on the heap, copy it recursively, not just the pointer
  - Remember to allocate memory while unpacking
- Sizing, Packing, and Unpacking must scan the same variables in the same order
- Test PUP routines with “+balancer RotateLB”

Automatic Dynamic Load Balancing

- Measurement based load balancers
  - **Principle of persistence:** In many CSE applications, computational loads and communication patterns tend to persist, even in dynamic computations
  - So, recent past is a good predictor of near future
  - Charm++ provides a suite of load-balancers
  - periodic measurement and migration of objects
- Seed balancers (for task-parallelism)
  - Useful for divide-and-conquer and state-space-search applications
  - Seeds for charm++ objects moved around until they take root

Using the Load Balancer

- link a LB module
  - ***-module <strategy>***
  - RefineLB, NeighborLB, GreedyCommLB, others…
  - EveryLB will include all load balancing strategies
- compile time option (specify default balancer)
  - **-balancer RefineLB**
- runtime option
  - ***+balancer RefineLB***

The code

```
void Jacobi::attemptCompute() {
  if (ghostReceived == numGhosts) {
    .... do computation
    if (step & 0x0F == 0) AtSync();
    else ResumeFromSync();
  }
}

void Jacobi::ResumeFromSync() {
    CkCallback cb (CkIndex_Main::stepCheckin(CkReductionMsg*), mainProxy);
    contribute(sizeof(double), &maxDiff, CkReduction::max_double, cb);
}
```

Performance: processor util.