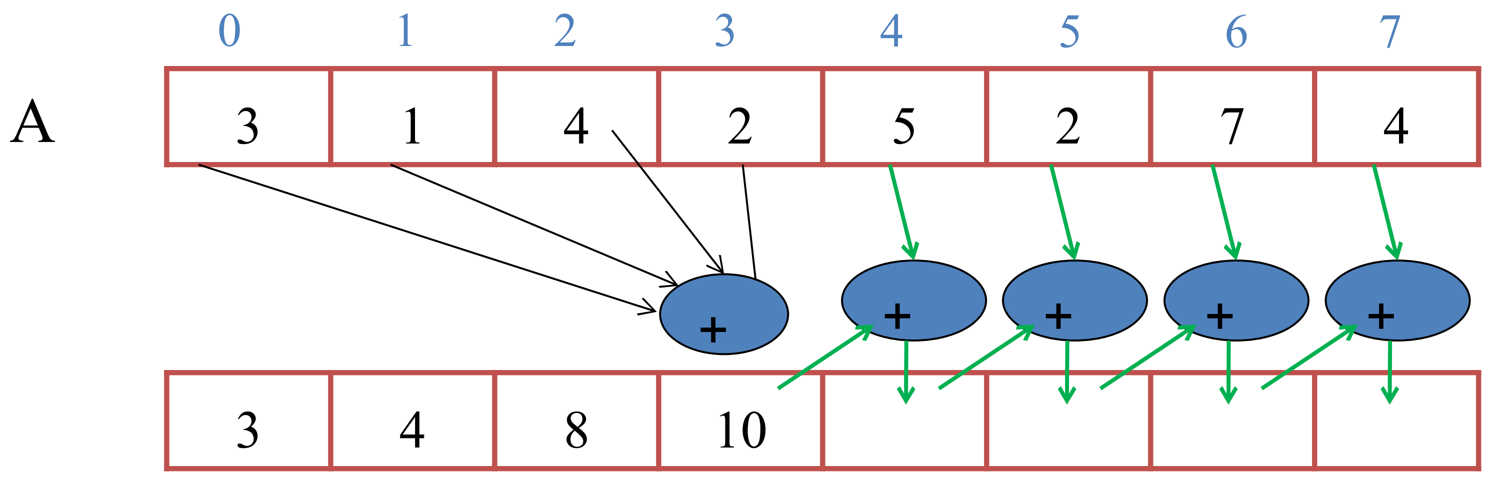
- Given array A[0..N-1], produce B[N], such that B[k] is the sum of all elements of A upto A[k]

Prefix Sum Problem



B[3] is the sum of A[0], A[1], A[2], A[3]

But B[3] can also be calculated as B[2]+ A[3]

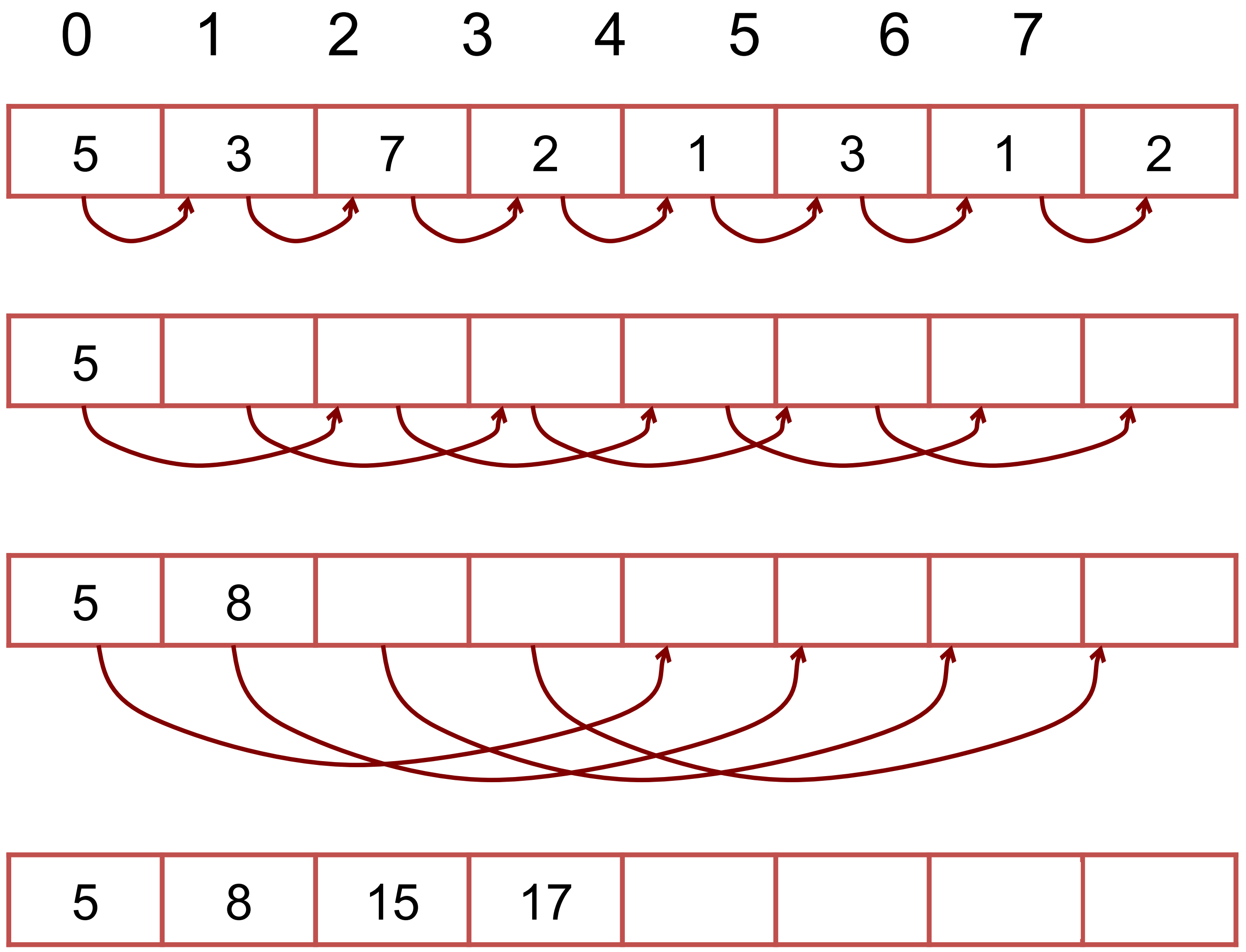
Parallel Prefix

- Data dependency from iteration to iteration.
  - How can this be parallelized at all?

```
B[0] = A[0];
for (i=1; i<N; i++)
  B[i] = B[i-1] + A[i];
```

- It looks like the problem is inherently sequential, but theoreticians came up with a beautiful algorithm called recursive doubling or just parallel prefix

Parallel prefix : recursive doubling



N Data Items

P Processors

N=P

Log P Phases

P additions in each phase

P log P operations

Completes in O(logP) time

Parallel Prefix Example: prefix.ci

```
mainmodule prefix {
    readonly CProxy_Main mainProxy;
    readonly CProxy_Prefix prefixArray;
    readonly int numElements;

    mainchare Main {
        entry Main(CkArgMsg* msg);
        entry [reductiontarget] void done();
    };

    array [1D] Prefix {
        entry Prefix();
        entry void step();
        entry void passValue(int value);
    }
}
```

```
#include "prefix.decl.h"
#include <math.h>
/*readonly*/ CProxy_Main mainProxy;
/*readonly*/ CProxy_Prefix prefixArray;
/*readonly*/ int numElements;

class Main : public CBase_Main {
public:
    Main(CkArgMsg *msg) {
        mainProxy = thisProxy;
        numElements = (msg->argc > 1) ? atoi(msg->argv[1]) :
8;
       delete msg;
        prefixArray = CProxy_Prefix::ckNew(numElements);
    }
    void done() { CkExit(); }
};
```

```
class Prefix : public CBase_Prefix {
    int value, distance;
public:
    Prefix() : distance(1) {
        srand(time(NULL));
        value = rand() % 10;
        step();
    }

    ...
    void step() {
        if (distance >= numElements) {
            CkPrintf("Prefix[%d].value = %d\n", thisIndex, value);
            CkCallback cb(CkReductionTarget(Main, done), mainProxy);
            contribute(sizeof(int), &value, CkReduction::sum_int, cb);
        }
        else {
            if (thisIndex+distance < numElements)
                 thisProxy[thisIndex + distance].passValue(value);
           }
        }
    }
    void passValue(int incoming_value) { ... }
};
#include "prefix.def.h"
```

```
void passValue(int
incoming_value) {
    value += incoming_value;
    distance *= 2;
    step();
}
```

```
    void step() {
        if (distance >= numElements) {
            CkPrintf("Prefix[%d].value = %d\n", thisIndex, value);
            CkCallback cb(CkReductionTarget(Main, done), mainProxy);
            contribute(sizeof(int), &value, CkReduction::sum_int, cb);
        }
        else {
            if (thisIndex+distance < numElements)
                 thisProxy[thisIndex + distance].passValue(value);
           }
            //if you no longer receive, but need to continue sending
            if (thisIndex - distance < 0) {
                 distance = distance*2;
                 step();
            }
        }
    }
    void passValue(int incoming_value) { ... }
};
#include "prefix.def.h"
```

```
void passValue(int
incoming_value) {
    value += incoming_value;
    distance *= 2;
    step();
}
```

```
    void step() {
        if (distance >= numElements) {
            CkPrintf("Prefix[%d].value = %d\n", thisIndex, value);
            CkCallback cb(CkReductionTarget(Main, done), mainProxy);
            contribute(sizeof(int), &value, CkReduction::sum_int, cb);
        }
        else {
            if (thisIndex+distance < numElements)
                thisProxy[thisIndex + distance].passValue(value);
            }
            //if you no longer receive, but need to continue sending
            if
                distance = distance*2;
                step();
            }
        }
    }
    void passValue(int incoming_value) { ... }
};
#include "prefix.def.h"
```

Still wrong Parallel Prefix: Why?

```
void passValue(int
incoming_value) {
    value += incoming_value;
    distance *= 2;
    step();
}
```

Parallel Prefix Example, Correct Version: prefix.ci

```
mainmodule prefix {
    readonly CProxy_Main mainProxy;
    readonly CProxy_Prefix prefixArray;
    readonly int numElements;
    readonly int numStages;

    mainchare Main {
        entry Main(CkArgMsg* msg);
        entry [reductiontarget] void done();
    };

    array [1D] Prefix {
        entry Prefix();
        entry void step();
        entry void passValue(int stage, int value);
    }
}
```

```
#include "prefix.decl.h"
#include <math.h>
/*readonly*/ CProxy_Main mainProxy;
/*readonly*/ CProxy_Prefix prefixArray;
/*readonly*/ int numElements;
/*readonly*/ int numStages;

class Main : public CBase_Main {
public:
    Main(CkArgMsg *msg) {
        mainProxy = thisProxy;
        numElements = (msg->argc > 1) ? atoi(msg->argv[1]) :
8;
        numStages = (int) ceil(log2(numElements));
       delete msg;
        prefixArray = CProxy_Prefix::ckNew(numElements);
    }
    void done() { CkExit(); }
};
```

```
class Prefix : public CBase_Prefix {
    int *flagBuf, *valueBuf, value, stage;
public:
    Prefix() : stage(0) {
        srand(time(NULL));
        value = rand() % 10;
        valueBuf = new int[numStages];
        flagBuf = new int[numStages];
        step();
    }

    ...
    void step() {
        if (stage >= numStages) {
            CkPrintf("Prefix[%d].value = %d\n",
            CkCallback cb(CkReductionTarget(Mair
            contribute(sizeof(int), &value, CkRe
        }
        else {
            int sendIndex = thisIndex + (1 << st
distance
            if (sendIndex < numElements)
                thisProxy[sendIndex].passValue(s

            if (flagBuf[stage] == 1)
                updateValue();
            else if (thisIndex - (1 << stage) <
                stage++;
                step();
            }
        }
    }
    ...
```

```
void passValue(int stg, int val)
{
    flagBuf[stg] = 1;
    valueBuf[stg] = val;
    if (flagBuf[stg] == 1)
        updateValue();
}

void updateValue() {
    value += valueBuf[stage];
    flagBuf[stage] = 0;
    stage++;
    step();
}
```

Parallel Prefix with SDAG: prefix.ci

```
mainmodule prefix {
    readonly CProxy_Main mainProxy;
    readonly CProxy_Prefix prefixArray;
    readonly int numElements;
    readonly int numStages;

    mainchare Main {
        entry Main(CkArgMsg* msg);
        entry [reductiontarget] void done();
    };

    array [1D] Prefix {
        entry Prefix();
        entry void passValue(int incoming_stage, int incoming_val);
        entry void step_through() { ... };
    }
}
```

```
entry void step_through() {
    for (stage = 0; stage < numStages; stage++) {
        serial "send_value" {
        int sendIndex = thisIndex + (1 << stage);
        if (sendIndex < numElements)
            thisProxy[sendIndex].passValue(stage, value);
        }

        if (thisIndex - (1 << stage) >= 0) {
            when passValue[stage](int stg, int val) {
                serial {
                    value += val;
                }
            }
        }
    }
    serial "done" {
        CkPrintf("Prefix[%d].value = %d\n", thisIndex,
value);
        CkCallback cb(CkReductionTarget(Main, done),
mainProxy);
        contribute(sizeof(int), &value, CkReduction::sum_int,
cb);
    }
    serial "done" {
        CkPrintf("Prefix[%d].value = %d\n", thisIndex,
value);
        CkCallback cb(CkReductionTarget(Main, done),
mainProxy);
        contribute(sizeof(int), &value, CkReduction::sum_int,
cb);
    }
};
```