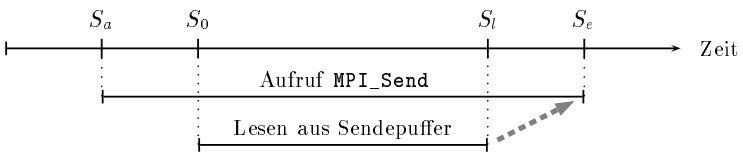
# 5 Punkt-zu-Punkt-Kommunikation in MPI

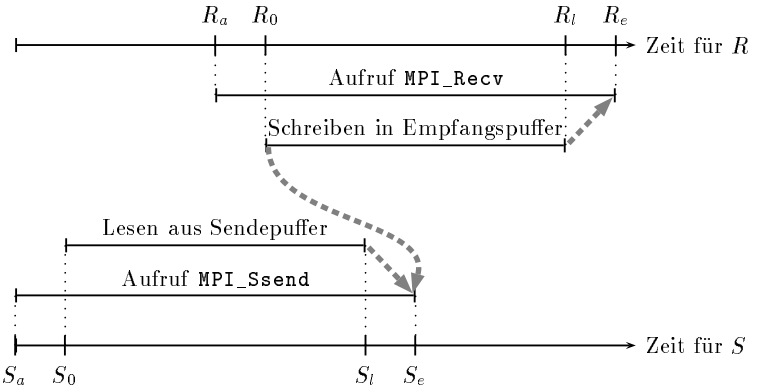
## 5.1 Blockierende Sendeoperationen




- MPI_Send kennen wir das wirklich schon?
- MPI_Ssend synchron
- MPI_Bsend „buffered“
- MPI_Rsend Empfänger ist „ready“
- MPI_Sendrecv
- MPI_Sendrecv_replace



### 5.1.1 Zyklische Datenverschiebung




Eine (bestenfalls) zweifelhafte Lösung:

```
MPI_S(s)end(sBuf, length, MPI_DOUBLE,
            iP+1+nP % nP, 7, MPI_COMM_WORLD);
MPI_Recv(rBuf, length, MPI_DOUBLE,
            iP-1+nP % nP, 7, MPI_COMM_WORLD, &st
```

$\Longrightarrow$ Verklemmung möglich (garantiert)



### 5.1.2 Handgestrickt: Eine komplizierte Lösung

```
if (iP & 1) {
  send();
  recv();
else {
  recv()
  send();
}
```

- $P$ gerade: geht so
- $P$ ungerade: langsam (Warum?)
- $P = 1$ Oops



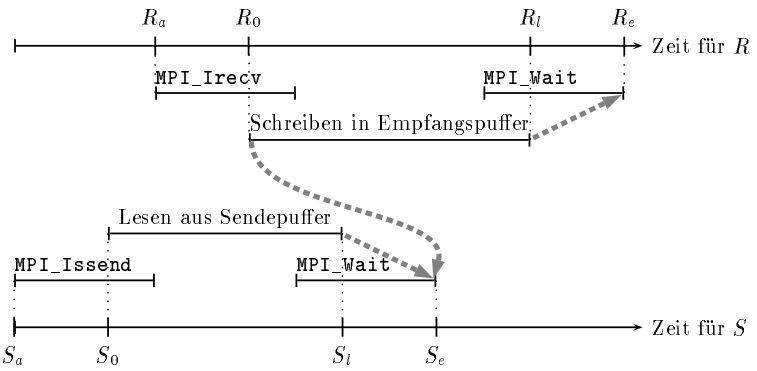
### 5.1.3 MPI_Bsend: Eine langsame Lösung

```
int size;
MPI_Pack_size(length, MPI_DOUBLE,
              MPI_COMM_WORLD, &size);
size += MPI_BSEND_OVERHEAD;
char *genug_speicher = malloc(size);
Assert(genug_speicher != 0);
MPI_Buffer_attach(genug_speicher, size);
...
MPI_Bsend(sBuf, length, MPI_DOUBLE,
          iP+1+nP % nP, 7, MPI_COMM_WORLD);
MPI_Recv(rBuf, length, MPI_DOUBLE,
          iP-1+nP % nP, 7, MPI_COMM_WORLD, &stat
...
int dummy;
char *dummyP;
MPI_Buffer_detach(&dummyP, &dummy);
```

## 5.2 Nichtblockierende Kommunikation

- Orthogonaler Satz von Sendeoperationen: `MPI_Isend`, `MPI_Issend`, `MPI_Ibsend` (sinnvoll?), `MPI_Irsend`
- Nichtblockierendes Empfangen: `MPI_Irecv`
- send bzw. receive stößt die Kommunikation an und bekommt ein `MPI_Request` zurück.
- Warten auf Komplettierung mit `MPI_Wait`. (u.v.a.m.)








### 5.2.1 Beispiel: Selbstbau von Sendrecv

```
void my_Sendrecv(void *sb, int sc, MPI_Datatype st,
                 int dst, int stag,
                 void *rb, int rc, MPI_Datatype rt,
                 int src, int rtag,
                 MPI_Comm comm, MPI_Status *status)
{ MPI_Status localStatus[2];
  MPI_Request req[2];

  MPI_Isend(sb, sc, st, dst, stag, comm, req + 0);
  MPI_Irecv(rb, rc, rt, src, rtag, comm, req + 1);
  MPI_Waitall(2, req, localStatus);
  *status = localStatus[1];
}
```



## 5.3 Weitere nichblockierende Funktionen

- `MPI_Waitany`, `MPI_Waitsome` fair!
- `MPI_Test`, `MPI_Testall`, `MPI_Testany`, `MPI_Testsome`
- Persistente requests: `MPI_Send_init`, ... `MPI_Start`

### 5.3.1 Nichtblockierende Kommunikation Sinnvoll?

– Komplizierter

– U.u. größerer overhead (mehr Funktionsaufrufe, ... )

\+ Deadlockvermeidung

\+ u.U. Höhere Bandbreite durch bidirektionale (+multiported?) Kommunikation

• Überlappung von Kommunikation und Berechnung (Kommunikationszeitverdeckung, latency hiding) Aber: U.u. niedrigere Bandbreite bei bestimmten Implementierungen (IBM SP?)



### 5.3.2 Kommunikationszeitverdeckung zur Zellularautomatensimulation

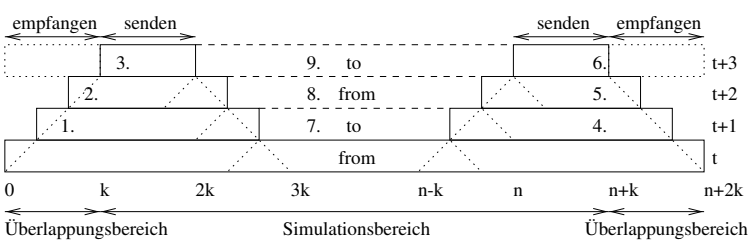
Hauptschleife:

- Simuliere Zeilen 1 und $n$
- Starte Kommunikation für die nächste Iteration
- Simuliere Zeilen 2 bis $n-1$
- Warte auf Komplettierung der der Kommunikation



### 5.3.3 Entkopplung + Überlappung

**PROCEDURE** simulate($x$, $y$, $k$) (* $k$ odd *)
**for** $i := k-1$ **downto** 0 **do**
  simulate lines $k-i$ through $2k+i$, $x \to y$
  simulate lines $n-i$ through $n+k+i$, $x \to y$
  swap $x$ and $y$
start send lines $x[k]$ through $x[2k-1]$ up
start send lines $x[n]$ through $x[n+k-1]$ down
start receive lines $y[0]$ through $y[k-1]$ from above
start receive lines $y[n+k]$ through $y[n+2k-1]$ from belo
**for** $i := 0$ **to** $k-1$ **do**
  simulate lines $3k-i-1$ through $n-k+i$, $y \to x$
  swap $x$ and $y$
complete communication






## Blick über den Tellerrand

- Anwendungen
  - Stömungssimulation, Spin-Gläser, Bildverarbeitung
  - Explizite iterative Lösung diskretisierter partieller Differentialgleichungen
  - Zeitabhängige partielle Differentialgleichungen
  - Partikelsimulation
- Weitere Algorithmische Probleme
  - Lastverteilung
  - Komplizierte Randbedingungen
  - Unregelmäßige Gitter, Schrittweiten und Überführungsfunktionen
  - Variable Gitter
  - Pyramiden bzw. Multigrid

# 6 Kollektive Operationen

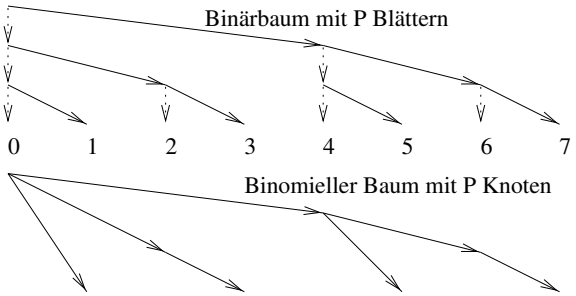
Wir kennen bereits:

- `MPI_Barrier`
- `MPI_Gather`



## 6.1 Broadcast: Implementierung

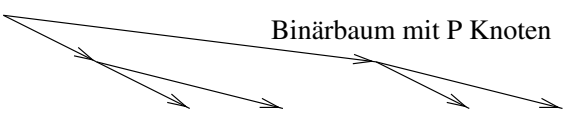
- root=0
- $P = 2^k$ am einfachsten
- kurze Nachrichten
- $T_{\text{broadcast}}(P, n) \approx (T_{\text{start}} + nT_{\text{byte}}) \log P$
- $P - 1$ Nachrichten a $n$ Byte
- Genaueres Modell (LogP ... ) führt zu leicht modifiziertem Kommunikationsmuster. (ausprobieren?)
- Multicast?





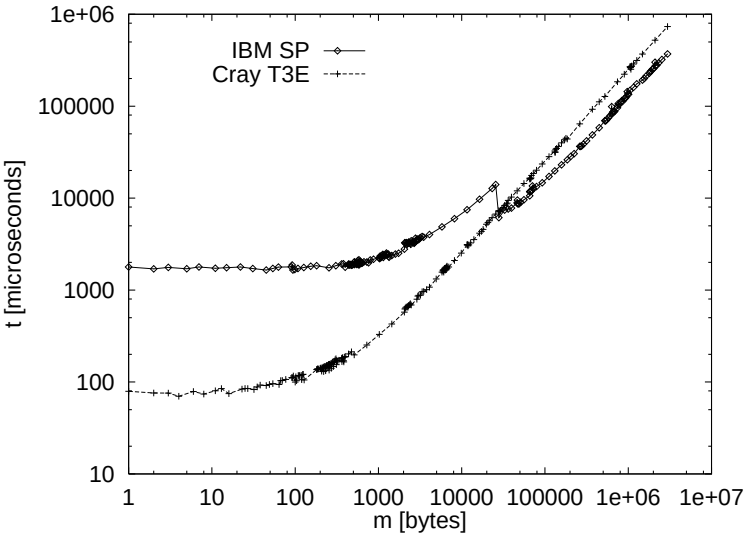

### 6.1.1 Pipelined Broadcast

- $P = 2^k - 1$
- **lange** Nachrichten
- $T_{\text{broadcast}}(P, n) \approx 3nT_{\text{byte}} + AT_{\text{start}} \log P$, $A \approx 3$–5?
- $P - 1$ Nachrichten a $n$ Byte
- Genaueres Modell (LogP, ... ) führt zu leicht modifiziertem Kommunikationsmuster. (ausprobieren?)
- **extrem lange** Nachrichten?






## 6.2 Reduktion

- $\otimes$ sei assoziativer (optional zus. kommutativer) Operator.
- Berechne $\bigotimes_{i<P} x_i$, wobei $x_i$ ein auf PE $i$ gelegener Vektor der Länge $n$ ist.
- Pipelined tree etc.: $T_{\text{reduce}} = \text{O}(nT_{\text{byte}} + T_{\text{start}} \log P)$
- Viele vordefinierte Operatoren. Genug?

### 6.2.1 IBM SP versus Cray T3E