The Collect Problem

- Collect up-to-date values of active processes
- A simple solution uses an array with $N$ entries
  \[ O(N) \] step complexity

But …
- $N$ is large
- Often only few of the processes take steps

Adaptive Collect Algorithms

- [Attiya, Fouren & Gafni, 1997] $O(K)$ (total contention)
- [Afek, Stupp & Touitou, 1999] $O(k^4)$
- [Attiya & Fouren, 2003] $O(k^2)$

But local step complexity depends on the total contention!

- [Afek, Stupp & Touitou, 2000] $O(k^4)$ local + shared
  - This work $O(k) / O(k^2)$ local + shared
  - For a generalized problem
**Collect & f**

- Compute a function $f$ on the stored values
  - associative, commutative and idempotent
  - $\text{put}&f(val)$, $\text{collect}&f()$
- A weaker variant: $\text{gather}&f$
  - Earlier gather may be more up-to-date than a later gather

**Implementing Gather & f / Collect & f**

- Naïve method: Use a gather / collect object
  - Local step complexity is not adaptive since $f$ is applied to all stored values
- Better method: Incrementally calculate the result as values are stored, and save it for later use
  - Extend idea from [Afek, Stupp & Touitou, 2000]
  - Fit into [Attiya and Fouren, 2003]

**Two Examples**

- **Pile** maintains the maximum value stored
  - $[\text{Afek et al. 2000}]$ $O(k^4)$
  - using our $\text{gather}&\text{max}$ $O(k^2)$
- **Active set** returns the current set of active processes
  - $[\text{Afek et al. 2000}]$ $O(k^4)$
  - using our $\text{gather}&\text{union}$ $O(k^2)$

**Repetetive Collect & f**

- Many algorithms invoke many put / collect operations within one high-level operation
  - Amortize costs over all invocations
  - $\text{start} \quad \text{put} \quad \text{collect} \quad \text{collect} \quad \text{put} \quad \text{collect} \quad \text{end}$
  - $\text{put}&f$ a single step
  - $\text{collect}&f$ $O(k)$ steps
  - start and end $O(k^2)$ steps
Implementing Repetitive Collect&f

\[\text{start}\{
\text{sign-in (active set)}\}
\]
\[\text{put}\{\]
\[R_i = \text{value}\}
\[\text{collect}\{\]
\[\text{get active set}\]
\[\text{read regs of active proc.}\]
\[\text{read from Pile}\]
\[\text{merge the results}\]
\[\text{end}\{\]
\[\text{put last result in pile}\]
\[\text{sign-out (active set)}\}

<table>
<thead>
<tr>
<th>Pile</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td></td>
</tr>
<tr>
<td>R_2</td>
<td></td>
</tr>
<tr>
<td>R_N</td>
<td></td>
</tr>
<tr>
<td>Active set</td>
<td>3</td>
</tr>
<tr>
<td>Result</td>
<td></td>
</tr>
</tbody>
</table>

Implications

- Atomic snapshots
  - [Afek, Stupp & Touitou, 2000] \(O(k^4)\) local+shared
  - [Attiya & Fouren, 2003] \(O(k^3)\) shared only
  - This work \(O(k^2)\) local+shared

- Immediate snapshots
  - [Attiya, Fouren & Gafni, 1997] \(O(K^3)\) shared (total)
  - [Afek, Stupp & Touitou, 2000] \(O(k^5)\) local+shared
  - This work \(O(k^3)\) local+shared

- Long-lived \((2k-1)-\)renaming
  - [Attiya, Fouren & Gafni, 2003] \(O(k^3)\) local+shared
  - Using our work