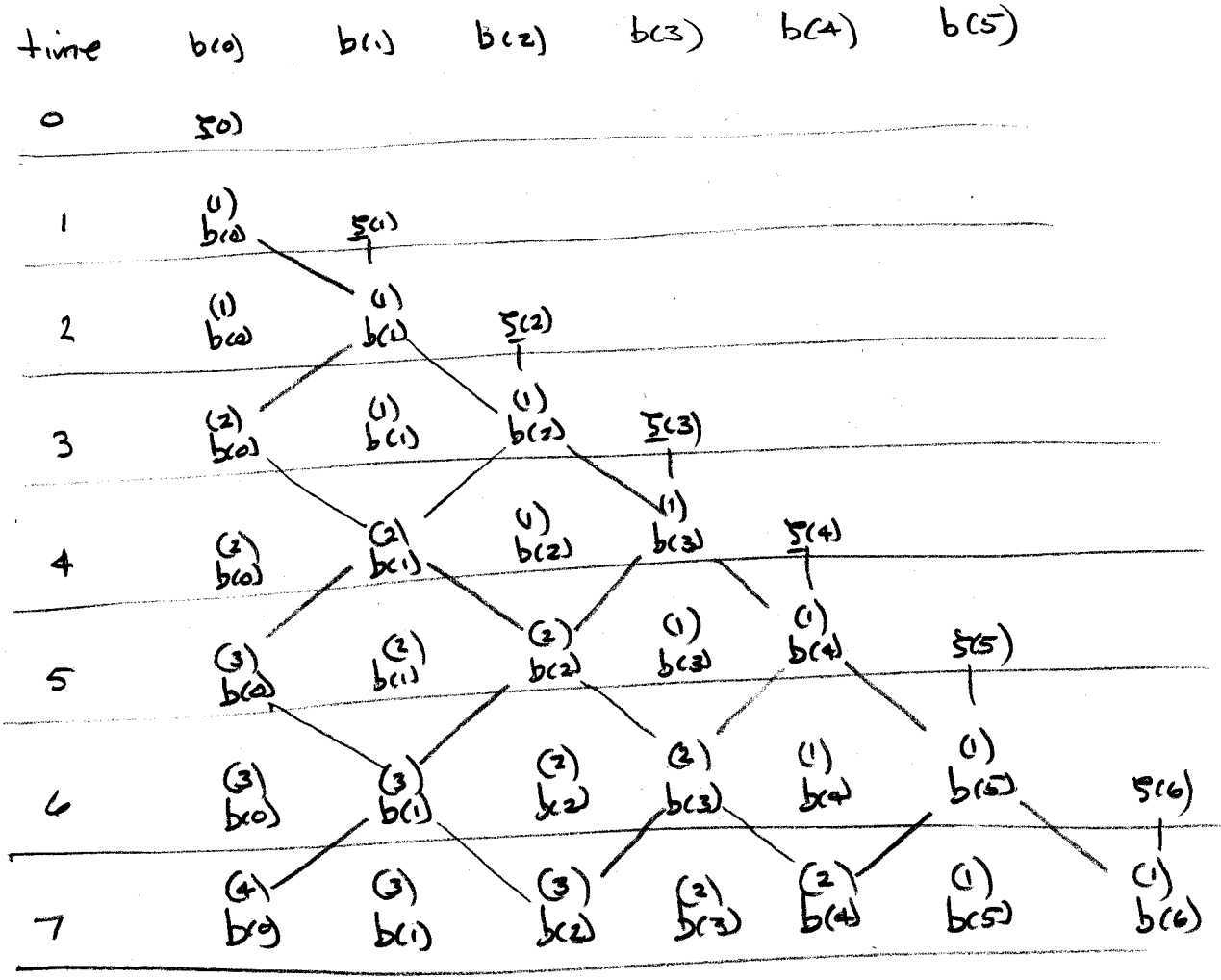
7.3 An Interference Cancellation Pipeline

- [Tan 00] showed that a regular structure can provide iterative solutions with a moderate delay that is independent of message length.
- For concreteness, assume G-S outer iteration and explicit matrix inversion for the inner calculation. We have

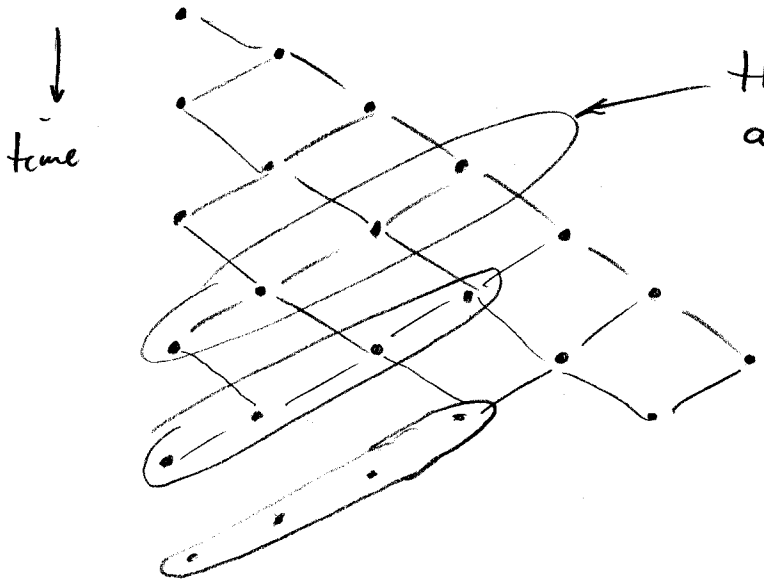
$$\underline{b}^{(j)}(n) = D_{Bnn}^{-1}\,\underline{s}(n) - L_{Bn,n-1}\,\underline{b}^{(j)}(n-1) - U_{Bn,n+1}\,\underline{b}^{(j-1)}(n+1)$$

We have at the end of each symbol interval.



- Single outer iteration per time step.
- New values at intersections
- Delay of $2J_0 - 1$ time steps for output to appear, then one per time step after that.

• To put it into a pipeline:



- Each unit is associated with an iteration number, and uses its own past vector plus the one from unit on the right to form

$$\underline{b}^{(j)}(n) = D_{Bnn}^{-1}\left(\underline{y}(n) - L_{Bn,n-1}\underline{b}^{(j)}(n-1) - U_{Bn,n+1}\underline{b}^{(j-1)}(n+1)\right)$$

Params $D_{Bnn}^{-1}\underline{y}(n)$, $D_{Bnn}^{-1}L_{Bn,n-1}$, $D_{Bnn}^{-1}U_{Bn,n+1}$ travel with $\underline{b}(n)$

- Units suggests delay of J_0 — but the one step compute time in each box makes it $2J_0 - 1$

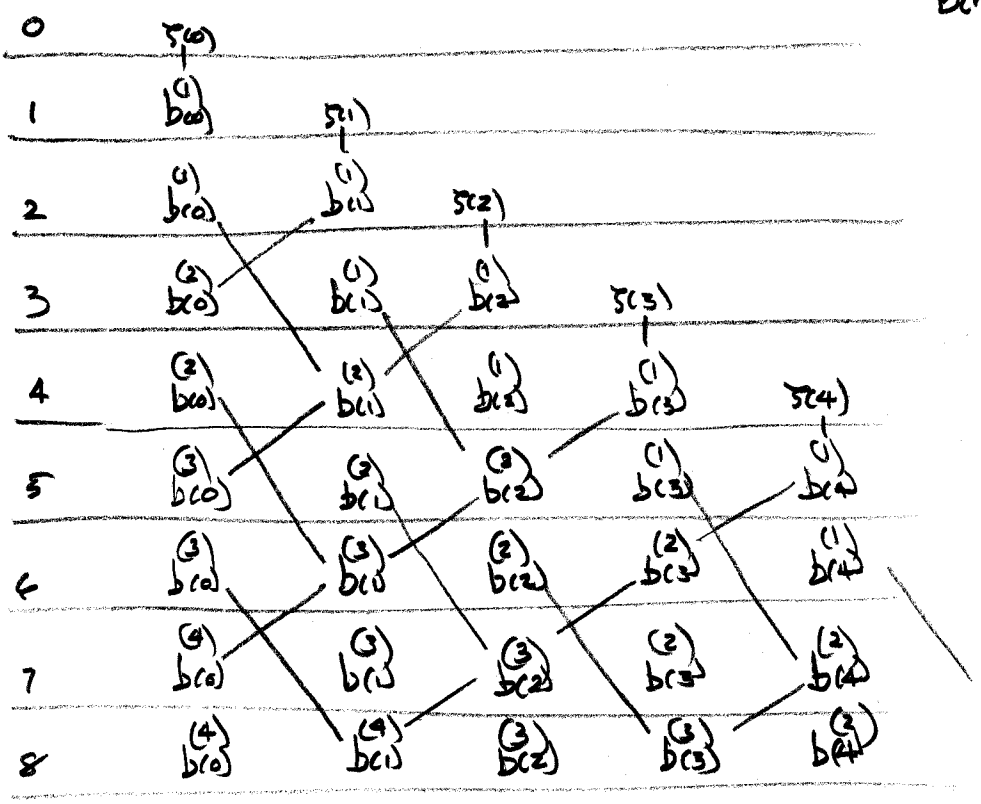
- The matrix block values themselves are formed progressively from MF/Rake/MRC

$$G = C^{+}\mathcal{S}CA$$

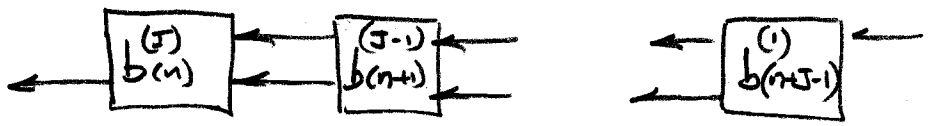
as the values come in.

- How about Jacobi outer iteration?

$$b^{(j)}(n) = f\left(b^{(j-1)}(n-1),\ b^{(j-1)}(n+1)\right)$$



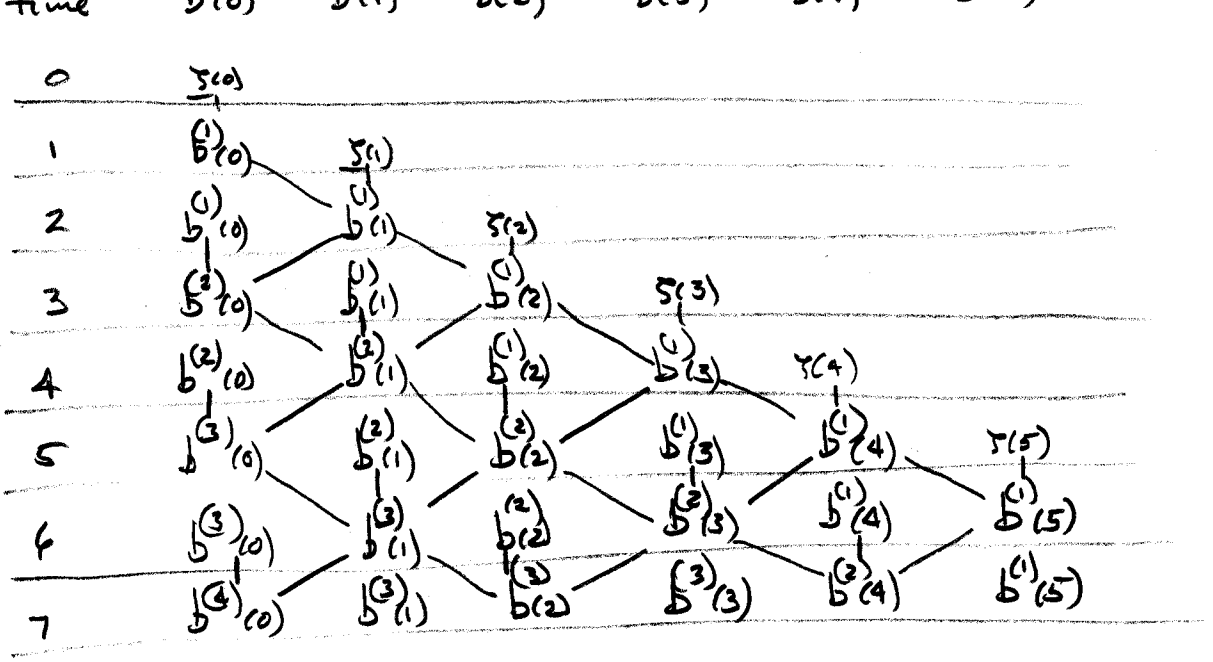
- Again the delay is $2J_0 - 1$, so no additional delay.
- Same hardware structure, unit associated with iteration



- Suppose $J = 3$. Leftmost unit receives $b^{(2)}(1)$ to form $b^{(3)}(0)$; two time steps later, uses it to form $b^{(3)}(2)$, so just hangs on to it.

- How about an iteration in which the update of a vector also depends on its own past value?

$$\underline{b}^{(j)}(n) = f\left(\underline{x}(n),\ \underline{b}^{(j)}(n-1),\ \underline{b}^{(j-1)}(n),\ \underline{b}^{(j-1)}(n+1)\right)$$



- Intermediate values are already passed left in previous time step, so unit just hangs onto them; e.g. $J=3$: leftmost unit uses $b^{(2)}(1)$ in forming both $b^{(3)}(0)$ and $b^{(3)}(1)$

- So same pipeline architecture, only the operations in the boxes differ

- The method can handle long codes as easily as short codes, although long codes require calculation of $L_{B_{n,n-1}}$, $D_{B_{n,n}}$, $U_{B_{n,n+1}}$ at each bit from the C and $\underline{S}$ knowledge (perfect CSI)

- We saw other limited-horizon methods in Section 4.3. In particular, [Jun98] suggests moving window, solution of the ZF or MMSE equations by CG (etc). Doing this bit by bit (user of symbol time n) is expensive. Block mode is more attractive. Pipeline?

- Next, we'll look at an alternative hardware organization for linear IC. As described in [Resm00], it applies to synchronous AWGN. Generalization not yet available. Synch, so block structuring not relevant.
- It operates on the sampled signal before MF. For simplicity, chip-synchronous, sample a chip-matched filter once per chip

Since synchronous,

$$\underline{r} = SA\underline{b} + \underline{n} \qquad \text{length-}N_c \text{ vectors.}$$

Cols of S are unit energy spreading sequences

$$S = [\underline{s}_1, \underline{s}_2, \cdots \underline{s}_K] \quad , \quad \underline{s}_i^+ \underline{s}_k = \delta_{ik}$$

- The calculations run like this:

user 1: $y_{1,1} = \underline{s}_1^+ \underline{r}$ component of $\underline{r}$ on $\underline{s}_1$ in 1st iter. estimate of $A_1 b_1$

$$\left.\begin{aligned} \underline{e}_{1,2} &= \underline{r} - \underline{s}_1 \underline{s}_1^+ \underline{r} \\ &= (I - \underline{s}_1 \underline{s}_1^+)\underline{r} \end{aligned}\right\} \text{ projection of } \underline{r} \text{ onto } \underline{s}_1^\perp$$

user 2: $y_{1,2} = \underline{s}_2^+ \underline{e}_{1,2}$

$$\underline{e}_{1,3} = \underline{e}_{1,2} - \underline{s}_2 \underline{s}_2^+ \underline{e}_{1,2}$$

etc, then repeat

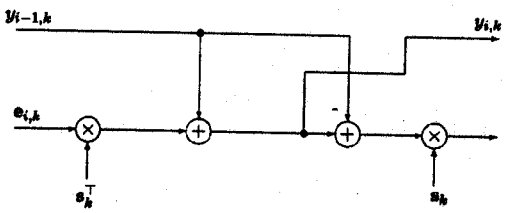


Fig. 1. Linear SIC unit.

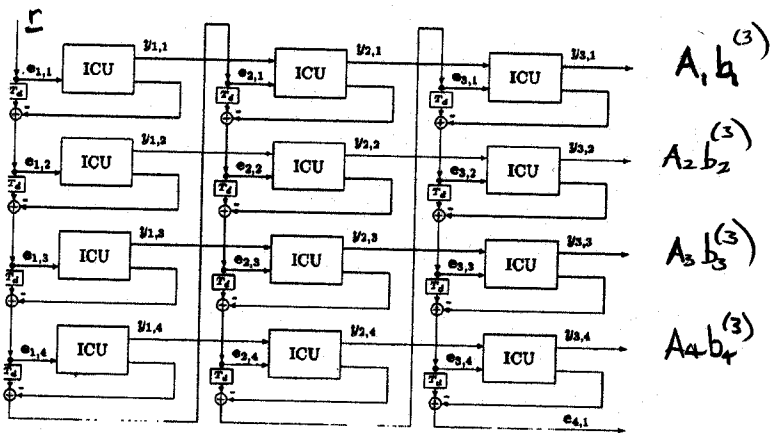


Fig. 2. Multistage linear SIC structure with four users and three stages.

Each unit subtracts a linear estimate of the component in the direction of its spreading code.

[Rasm 00] shows this is precisely the Gauss Seidel iteration.

- Advantages:
  - no explicit calculation of all the code inner products $R = S^{+}S$
  - nice for long codes
  - no explicit matrix inversion

Disadvantages
  - repeated inner products at the chip level.
  - each inner product must wait for its predecessors to complete, which hampers parallel processing, adds delay