Iterative Timing Recovery

John R. Barry

School of Electrical and Computer Engineering, Georgia Tech
Atlanta, Georgia U.S.A.
barry@ece.gatech.edu

Outline

Timing Recovery Tutorial

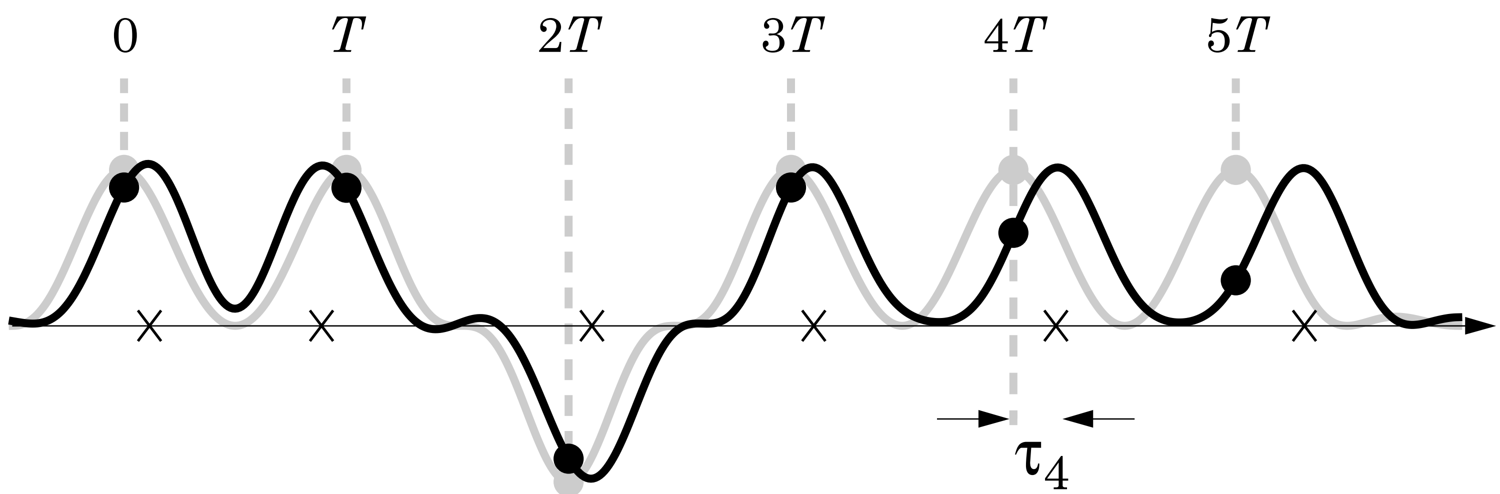
- Problem statement
- TED: M&M, LMS, S-curves
- PLL

Iterative Timing Recovery

- Motivation (powerful FEC)
- 3-way strategy
- Per-survivor strategy
- Performance comparison

The Timing Recovery Problem

Receiver expects the k-th pulse to arrive at time kT:

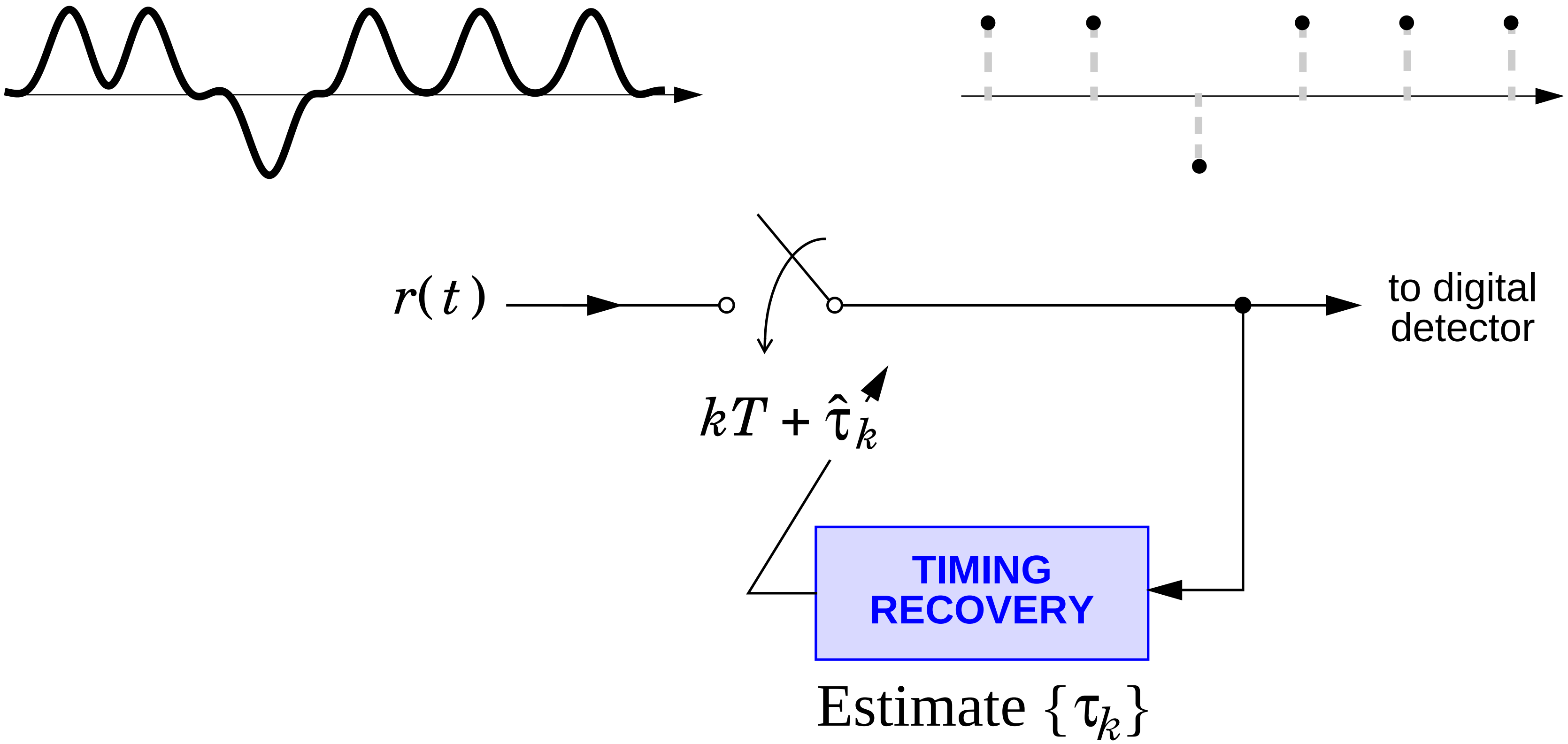


Instead, the k-th pulse arrives at time $kT + \tau_k$.

Notation: τ_k is *offset* of k-th pulse.

Sampling

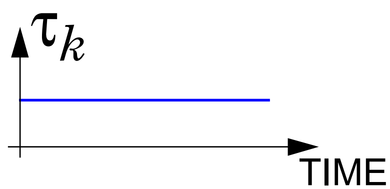
Best sampling times are $\{kT + \tau_k\}$.



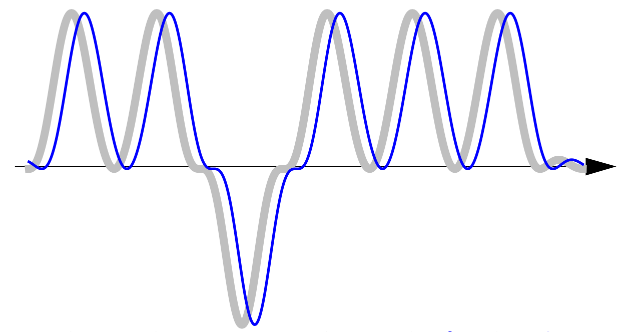
Timing Offset Models

CONSTANT

$$\tau_k = \tau_0$$

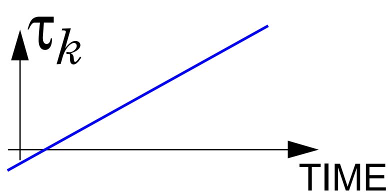


$\Rightarrow$

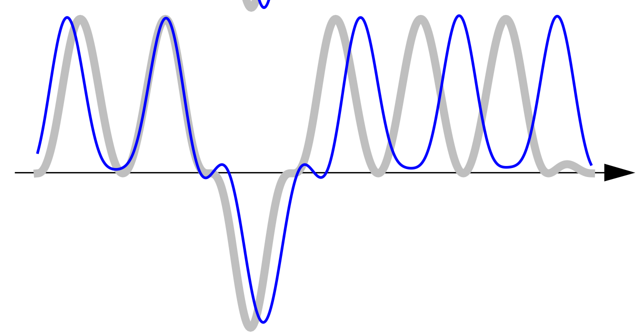


FREQUENCY OFFSET

$$\tau_{k+1} = \tau_k + \Delta T$$

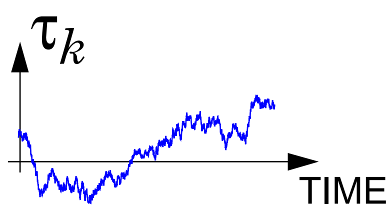


$\Rightarrow$

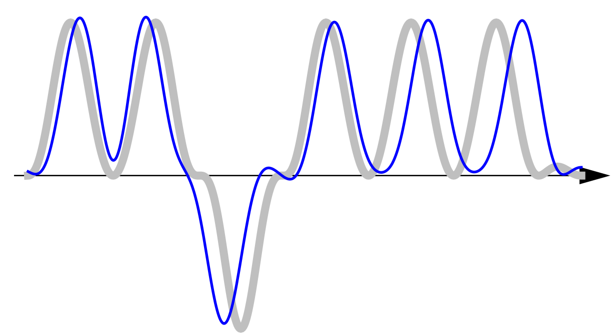


RANDOM WALK

$$\tau_{k+1} = \tau_k + \mathcal{N}(0, \sigma_w^2)$$



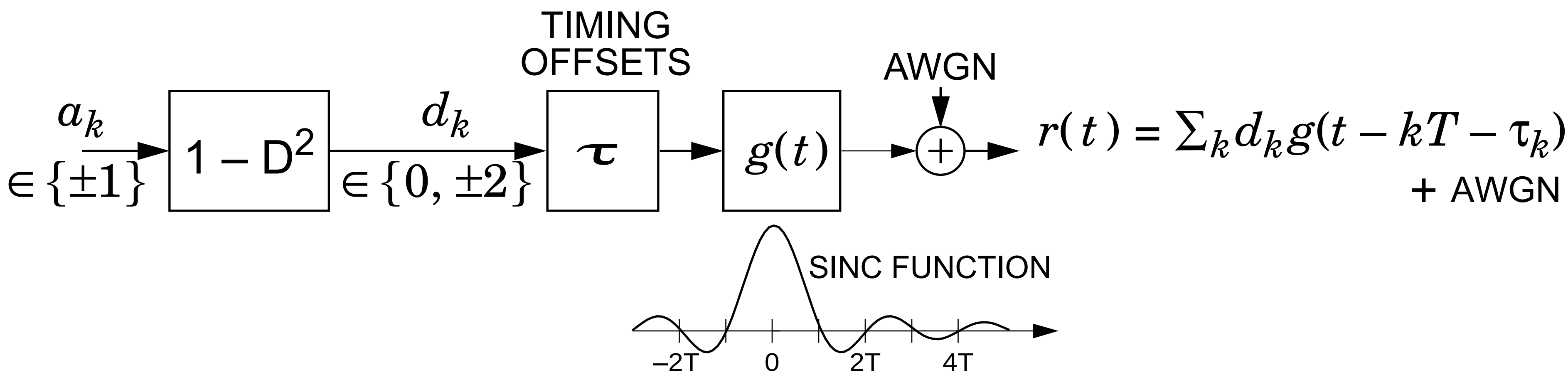
$\Rightarrow$



RANDOM WALK + FREQUENCY OFFSET

$$\tau_{k+1} = \tau_k + \mathcal{N}(\Delta T, \sigma_w^2)$$

The PR4 Model and Notation



Define:

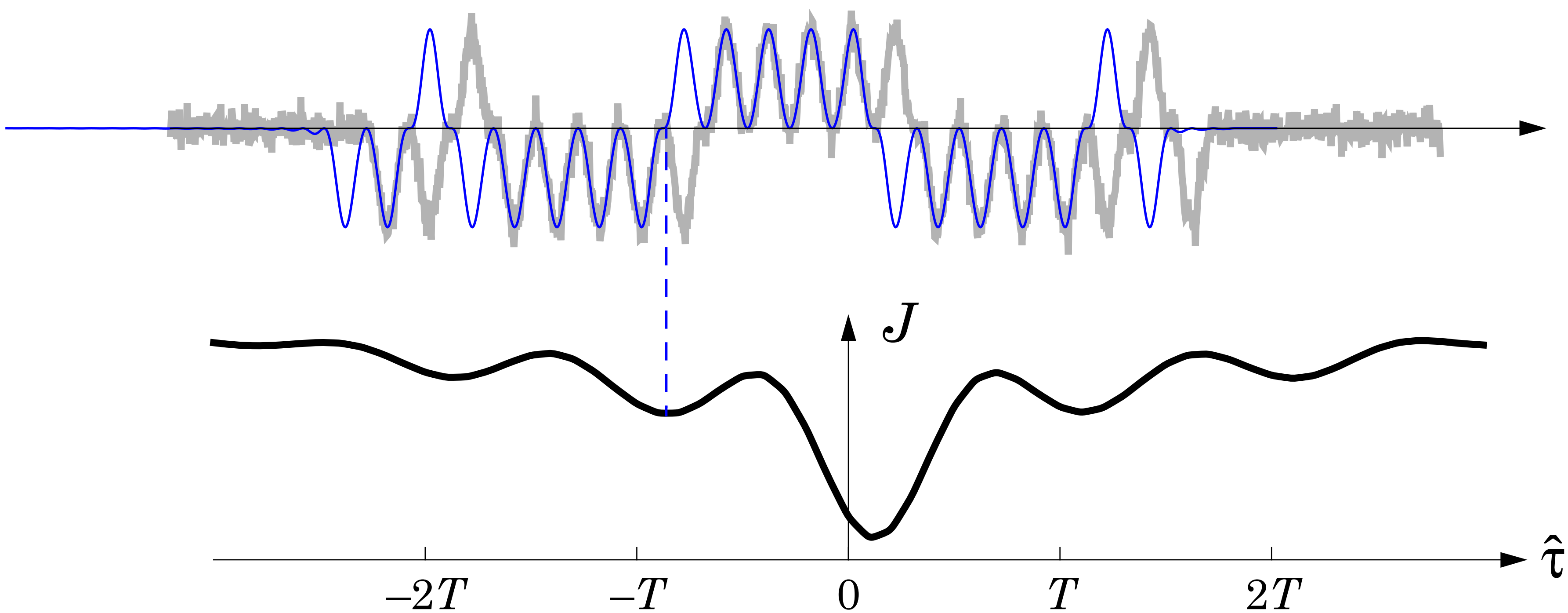
- $d_k = a_k - a_{k-2} \in \{0, \pm 2\}$ = 3-level "PR4" symbol
- $\hat{d}_k$ = receiver's estimate of d_k
- τ_k = timing offset
- $\hat{\tau}_k$ = receiver's estimate of τ_k
- $\varepsilon_k = \tau_k - \hat{\tau}_k$ estimate error, with std σ_ε.
- $\hat{\varepsilon}_k$ = receiver's estimate of ε_k

ML Estimate: Trained, Constant Offset

The ML estimate $\hat{\tau}$ minimizes $J(\tau \mid \boldsymbol{a}) = \int_{-\infty}^{\infty} \left(r(t) - \sum_i d_i g(t - iT - \tau) \right)^2 dt$.

Exhaustive search:

Try all values for τ, pick one that best represents $r(t)$ in MMSE sense.



ANIMATION 1

Achieves Cramer-Rao Bound

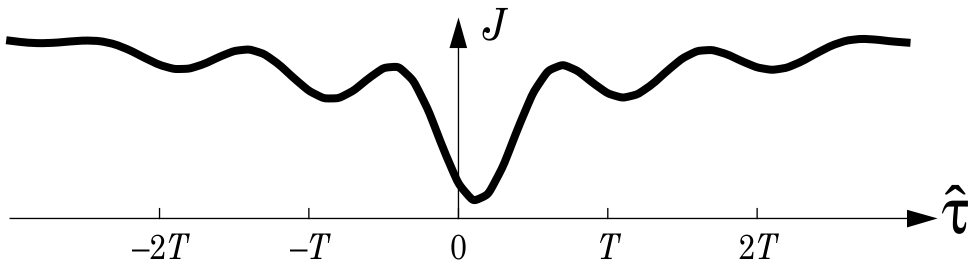
$$r(kT+\hat{\tau}) = \sum_i d_i g(kT - iT + \hat{\tau} - \tau) + n_k$$

$$= s_k(\varepsilon) + n_k, \quad \text{where } \varepsilon = \tau - \hat{\tau} \text{ is the estimation error.}$$

The CRB on the variance of the estimation error:

$$\frac{\sigma_\varepsilon^2}{T^2} \geq \frac{\sigma^2}{N} \cdot \frac{1}{E\left[\left(\frac{\partial}{\partial\varepsilon} s_k(\varepsilon)\right)^2\right]} = \frac{3\sigma^2}{\pi^2 N}.$$

Implementation



Gradient search: $$\hat{\tau}_{i+1} = \hat{\tau}_i - \mu \frac{\partial}{\partial \tau} J(\tau \mid \boldsymbol{a})_{\tau = \hat{\tau}_i}$$

Direct calculation of gradient:
$$\frac{1}{2}\frac{\partial}{\partial \tau} J(\tau \mid \boldsymbol{a}) = \sum_i d_i \int_{-\infty}^{\infty} r(t) g'(t - iT - \tau) dt$$
$$= \sum_i d_i r_i' \,.$$

Remarks:

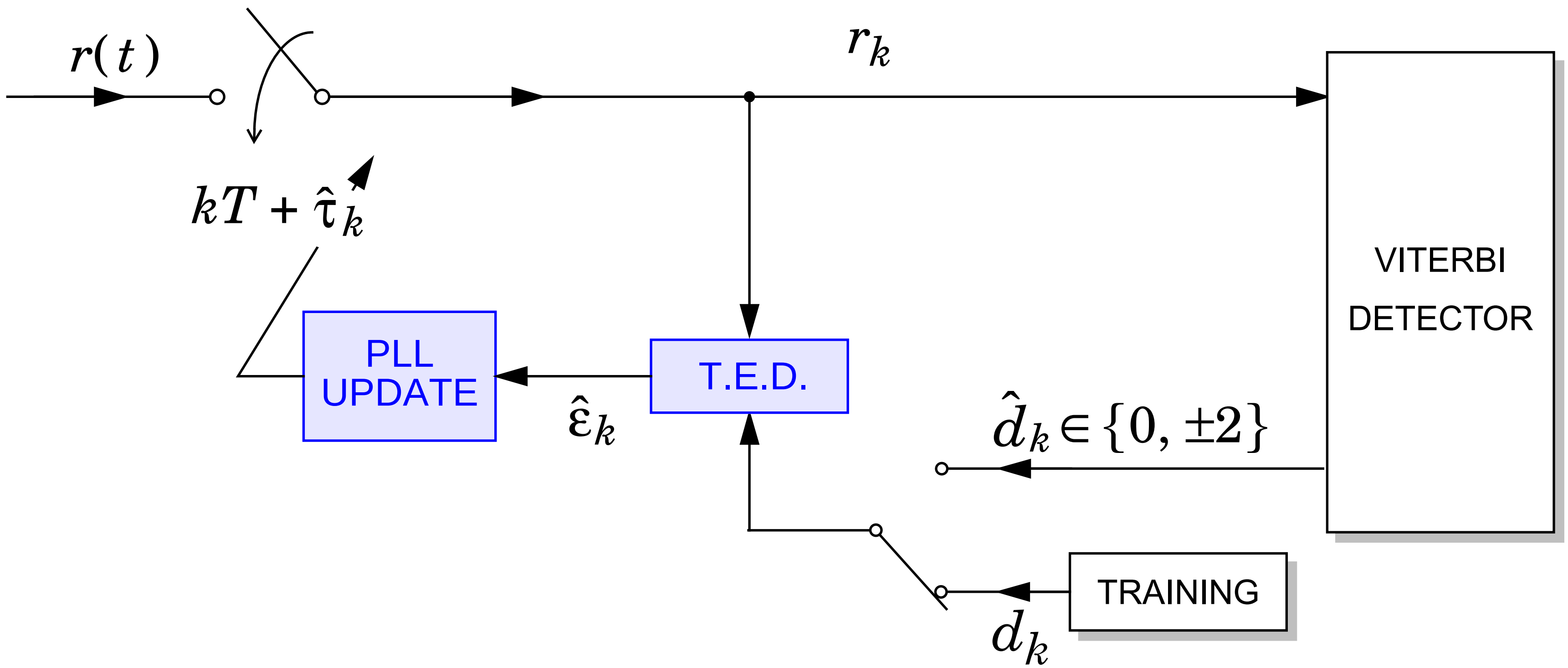
- Susceptible to local minima $\Rightarrow$ initialize carefully.
- Block processing.
- Requires training.

Conventional Timing Recovery

After each sample:

Step 1. Estimate residual error, using a timing-error detector (TED)

Step 2. Update $\hat{\tau}$, using a phase-locked loop (PLL)



LMS Timing Recovery

MMSE cost function:

$$E\left[\left(r_k - d_k\right)^2\right]$$

k-th sample,
$r_k = r(kT + \hat{\tau}_k)$

what we want it to be

LMS approach: $$\hat{\tau}_{k+1} = \hat{\tau}_k + \mu \hat{\varepsilon}_k$$

where $$\hat{\varepsilon}_k = -\frac{\partial}{\partial \hat{\tau}}\left(r_k - d_k\right)^2\Big]_{\hat{\tau} = \hat{\tau}_k}.$$

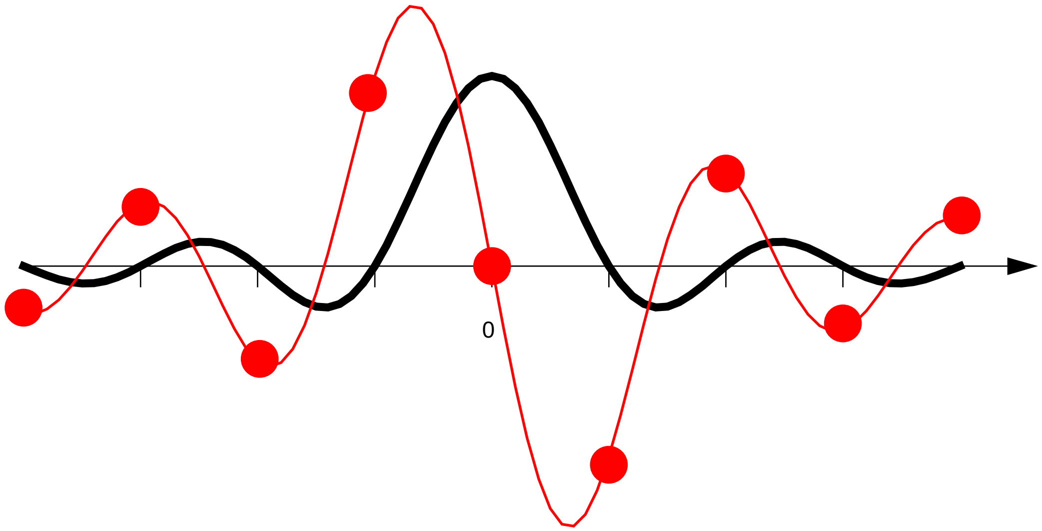
LMS TED

But: $\frac{1}{2}\frac{\partial}{\partial\hat{\tau}}\left(r_k - d_k\right)^2\Big]_{\hat{\tau} = \hat{\tau}_k} = (r_k - d_k)\frac{\partial}{\partial\hat{\tau}}\sum_i d_i g(kT - iT + \hat{\tau} - \tau)$

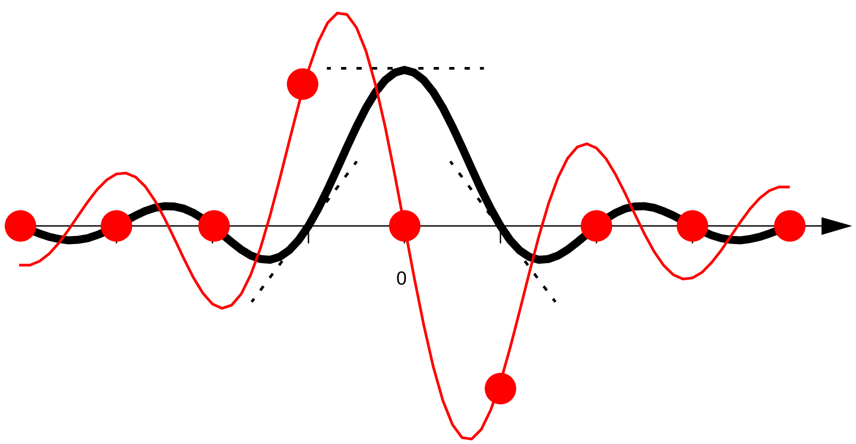
$$= (r_k - d_k)\sum_i d_i g'(kT - iT + \hat{\tau} - \tau)$$

$$= (r_k - d_k)\sum_i d_i p_{k-i}^{(\varepsilon_k)}$$

where $p_n^{(\varepsilon_k)} = \frac{\partial}{\partial\tau} g(nT - \varepsilon)$:



From LMS to Mueller & Müller



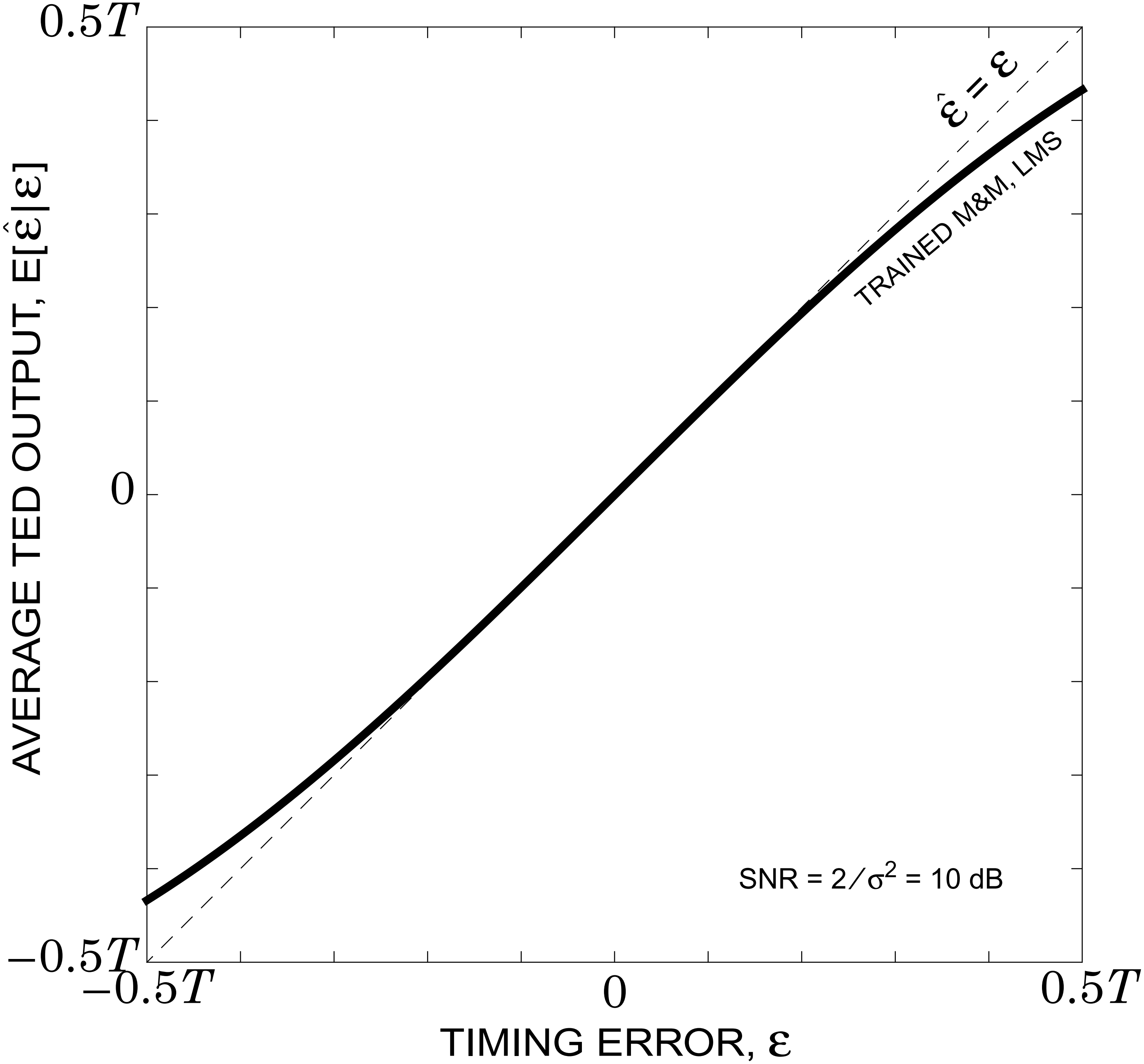
$$\hat{\varepsilon}_k \propto\approx (r_k - d_k)(d_{k-1} - d_{k+1}) + \text{smaller terms}$$

$$= r_k d_{k-1} - r_k d_{k+1} \underbrace{- d_k d_{k-1} + d_k d_{k+1}}_{\text{Independent of } \tau}$$

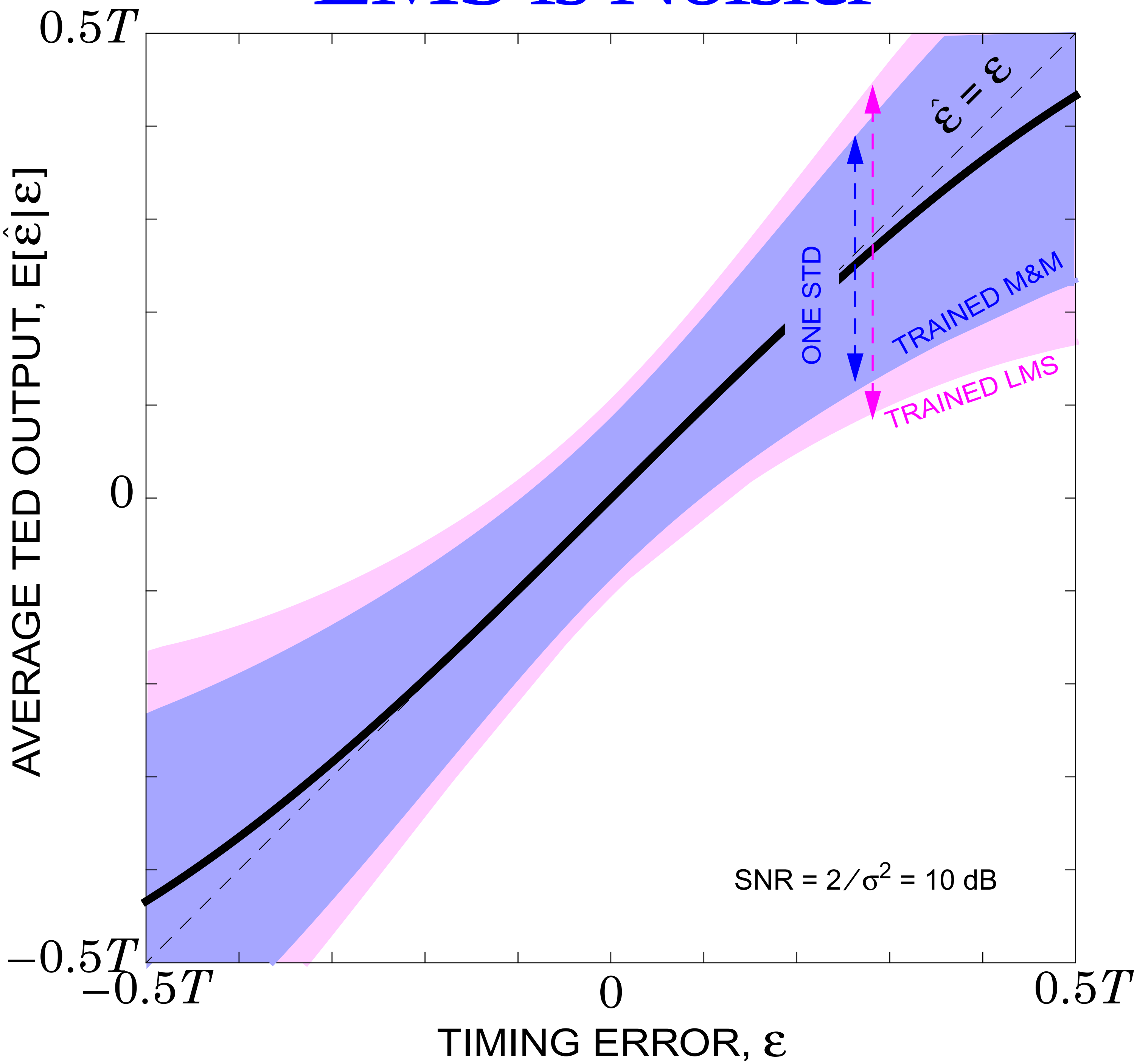
Delay second term and eliminate last two:

$$\hat{\varepsilon}_k \propto r_k d_{k-1} - r_{k-1} d_k \Rightarrow \text{ Mueller \& Müller (M\&M) TED}$$

S Curves

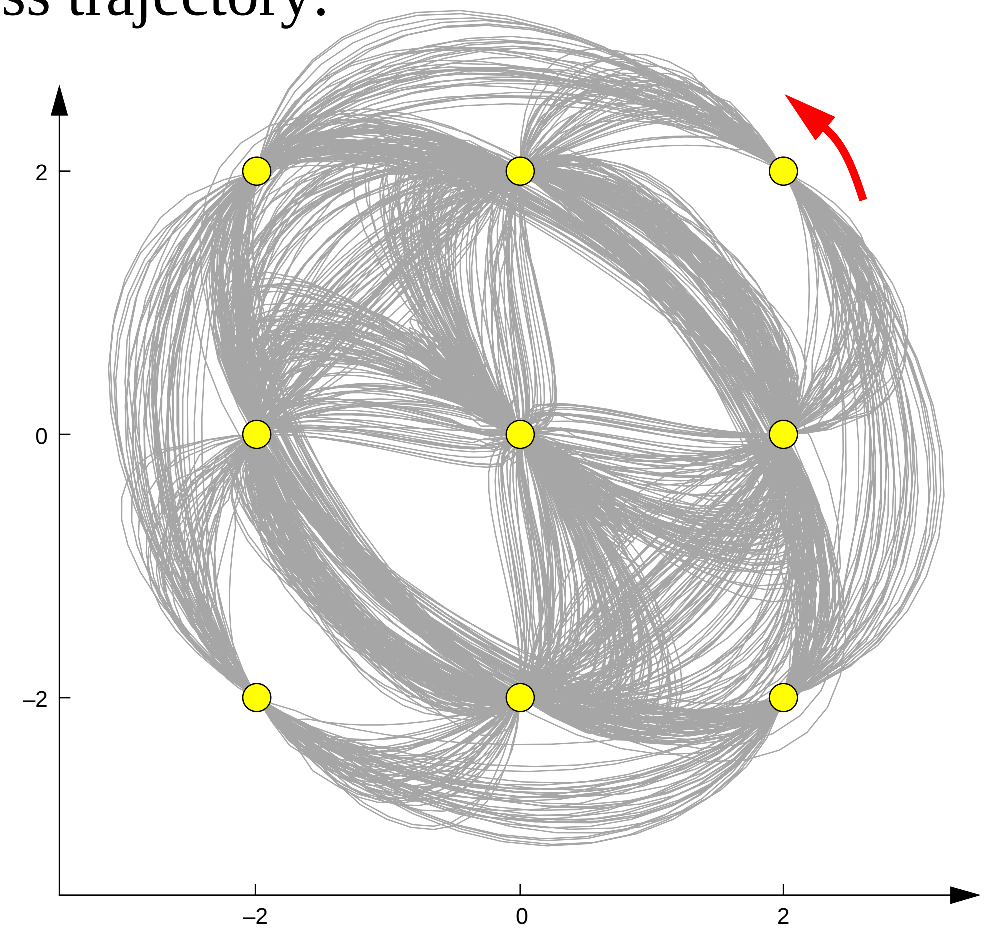

0.5T
0
−0.5T
−0.5T
0
0.5T
AVERAGE TED OUTPUT, E[ε̂|ε]
TIMING ERROR, ε
ε̂ = ε
TRAINED M&M, LMS
SNR = 2 ⁄ σ2 = 10 dB

LMS is Noisier



An Interpretation of M&M

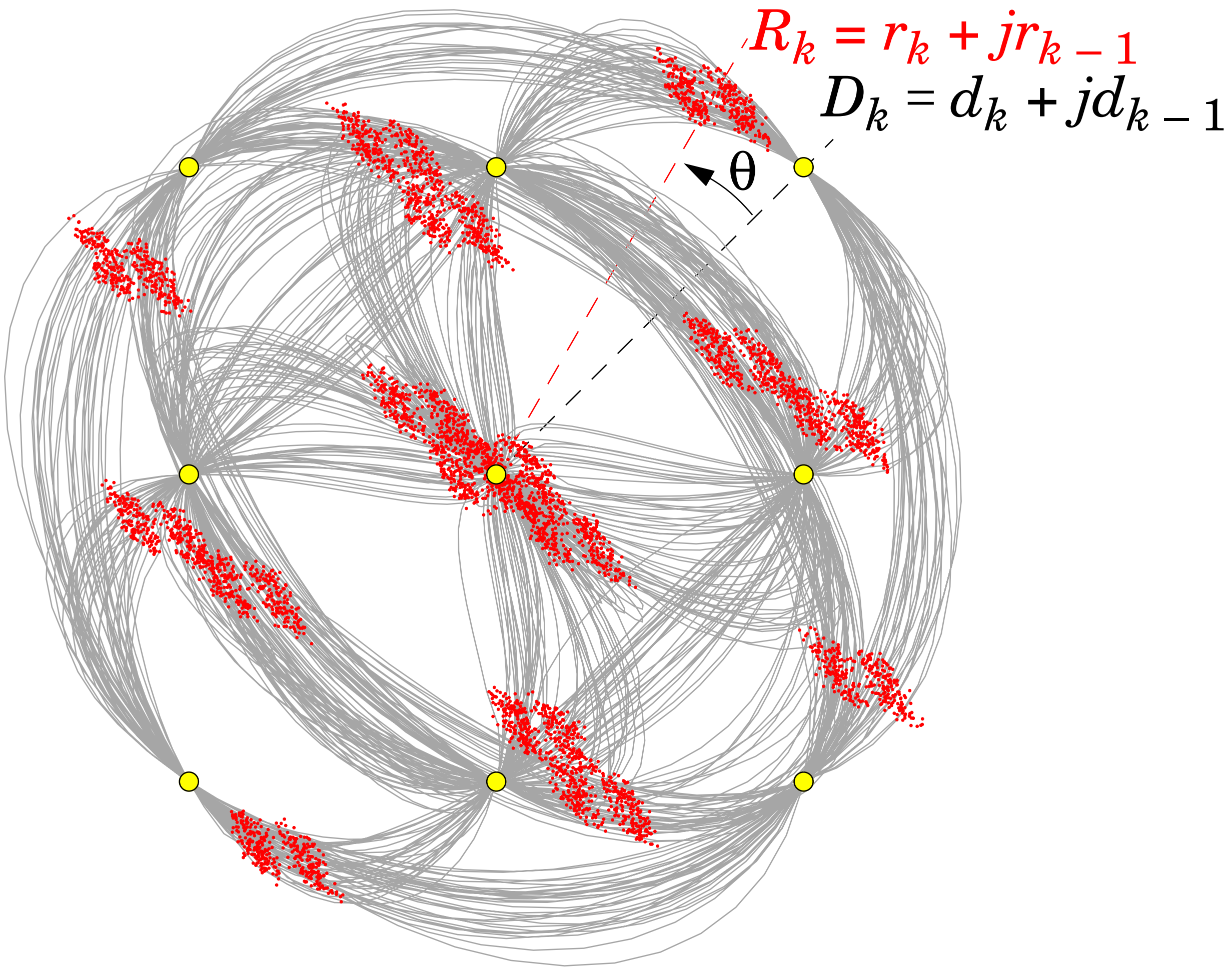
Consider the *complex* signal $r(t) + jr(t - T)$.
Its noiseless trajectory:



- It passes through $\{0, \pm 2, \pm 2 \pm 2j, \pm 2j\}$ at times $\{kT + \tau\}$.
- More often than not, in a counterclockwise direction.

ANIMATION 2

Sampling Late by 20%



The angle between R_k and D_k predicts timing error:

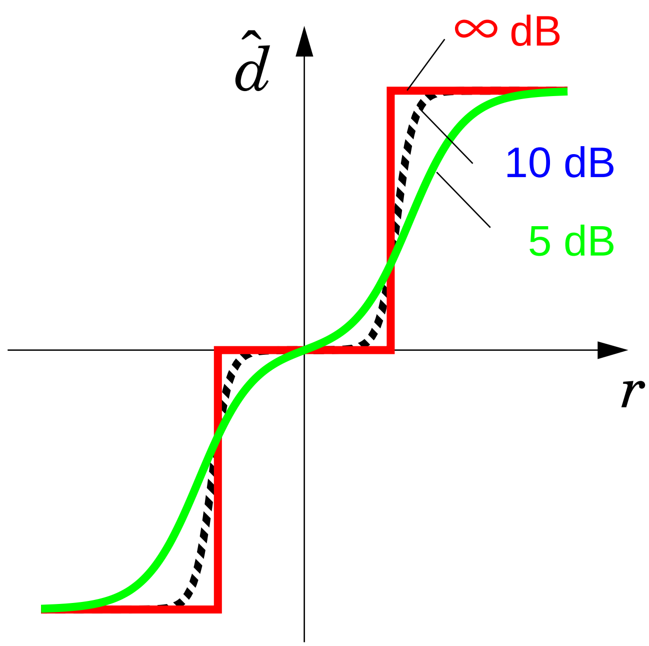
$$\theta \approx \sin\theta = \mathrm{Im}\left\{\frac{R^*D}{|RD|}\right\} \propto r_k d_{k-1} - r_{k-1} d_k \quad \Rightarrow \quad \text{M\&M.}$$

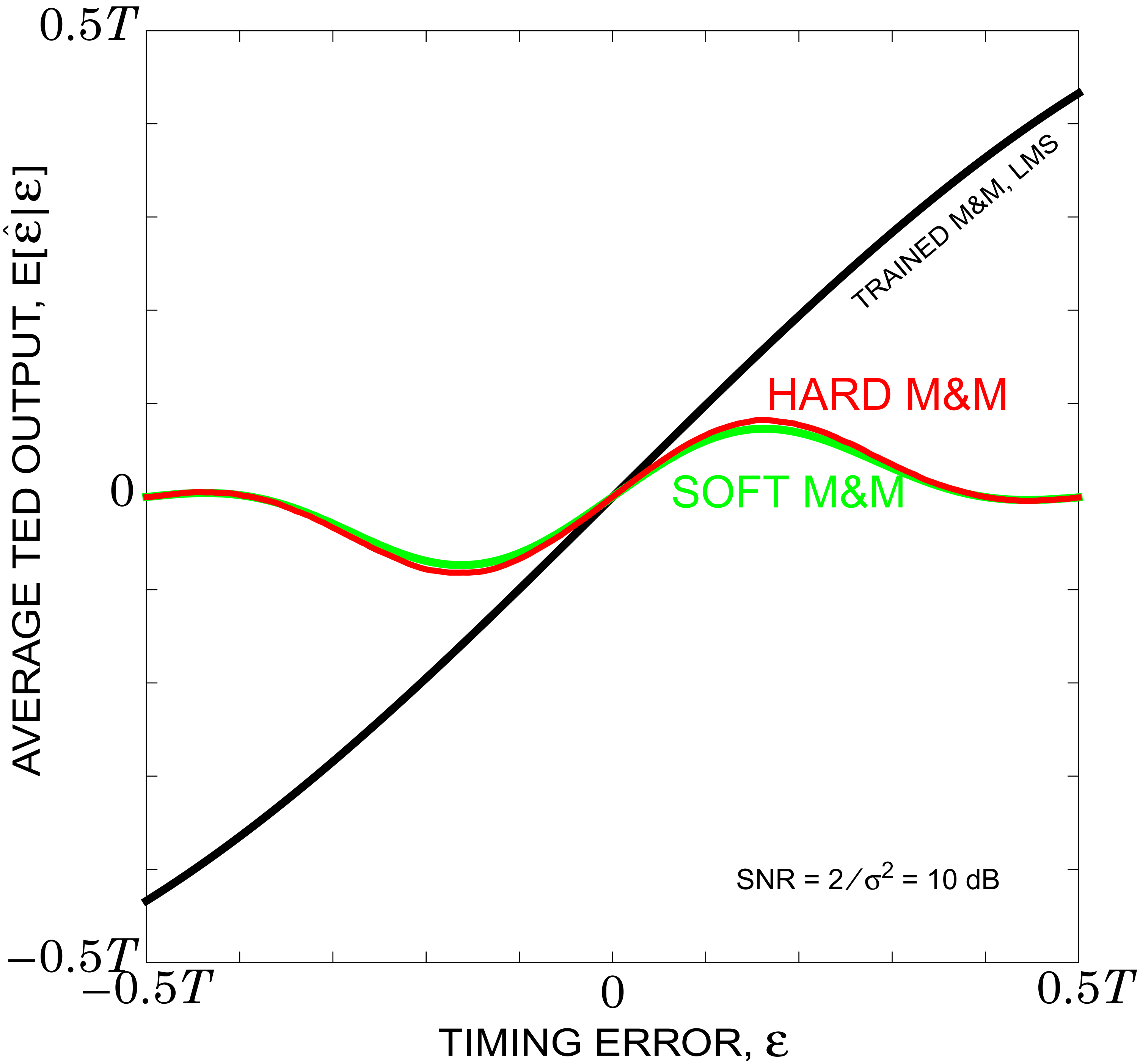
Decision-Directed TED

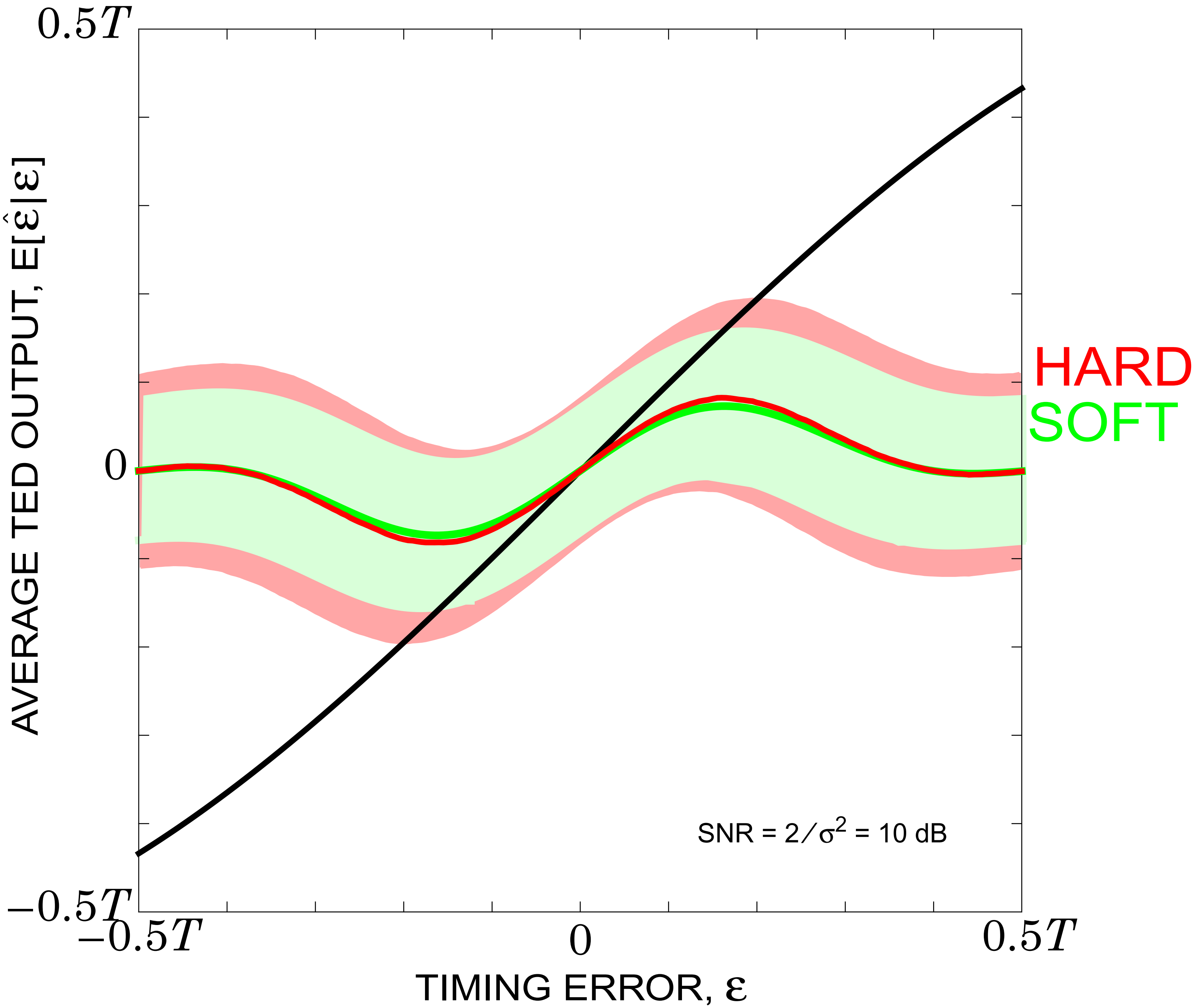
Replace training by decisions $\{\hat{d}_k\}$ $\Rightarrow$ $\hat{\varepsilon}_k \propto r_k\hat{d}_{k-1} - r_{k-1}\hat{d}_k$.

Instantaneous decisions:

- Hard: Round r_k to nearest symbol.
- Soft: $\hat{d}_k = \mathrm{E}[d_k \mid r_k] = \dfrac{2\sinh(2r_k/\sigma^2)}{\cosh(2r_k/\sigma^2) + e^{2/\sigma^2}}$:



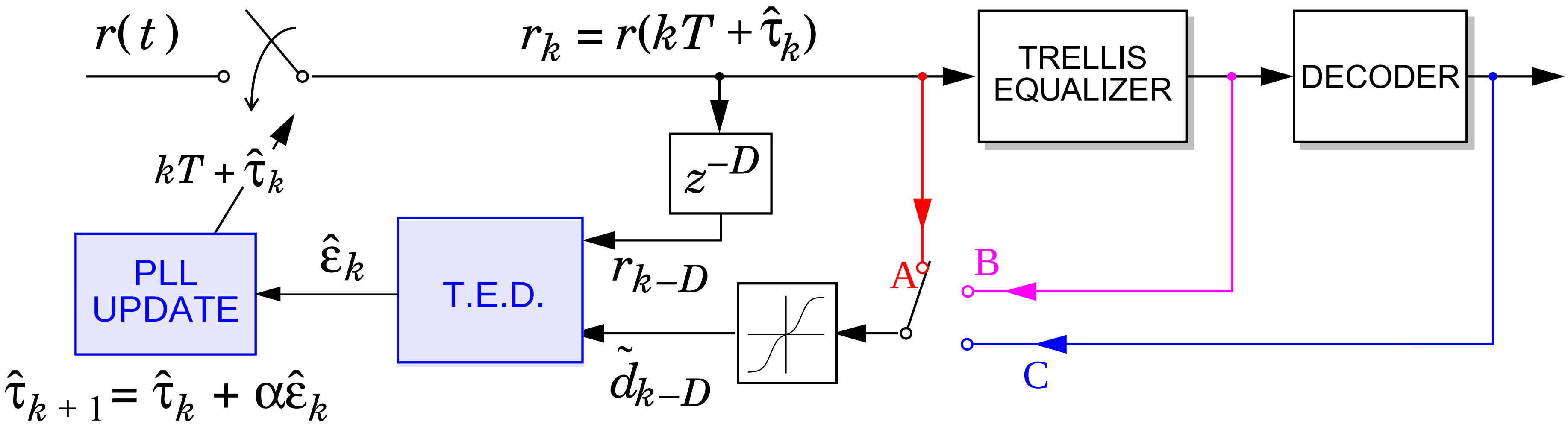

0.5T
0
−0.5T
−0.5T
0
0.5T
AVERAGE TED OUTPUT, E[ε̂|ε]
TIMING ERROR, ε
TRAINED M&M, LMS
HARD M&M
SOFT M&M
SNR = 2/σ2 = 10 dB



0.5T
0
−0.5T
−0.5T
0
0.5T
AVERAGE TED OUTPUT, E[ε̂|ε]
TIMING ERROR, ε
HARD
SOFT
SNR = 2/σ² = 10 dB

Reliability versus Delay

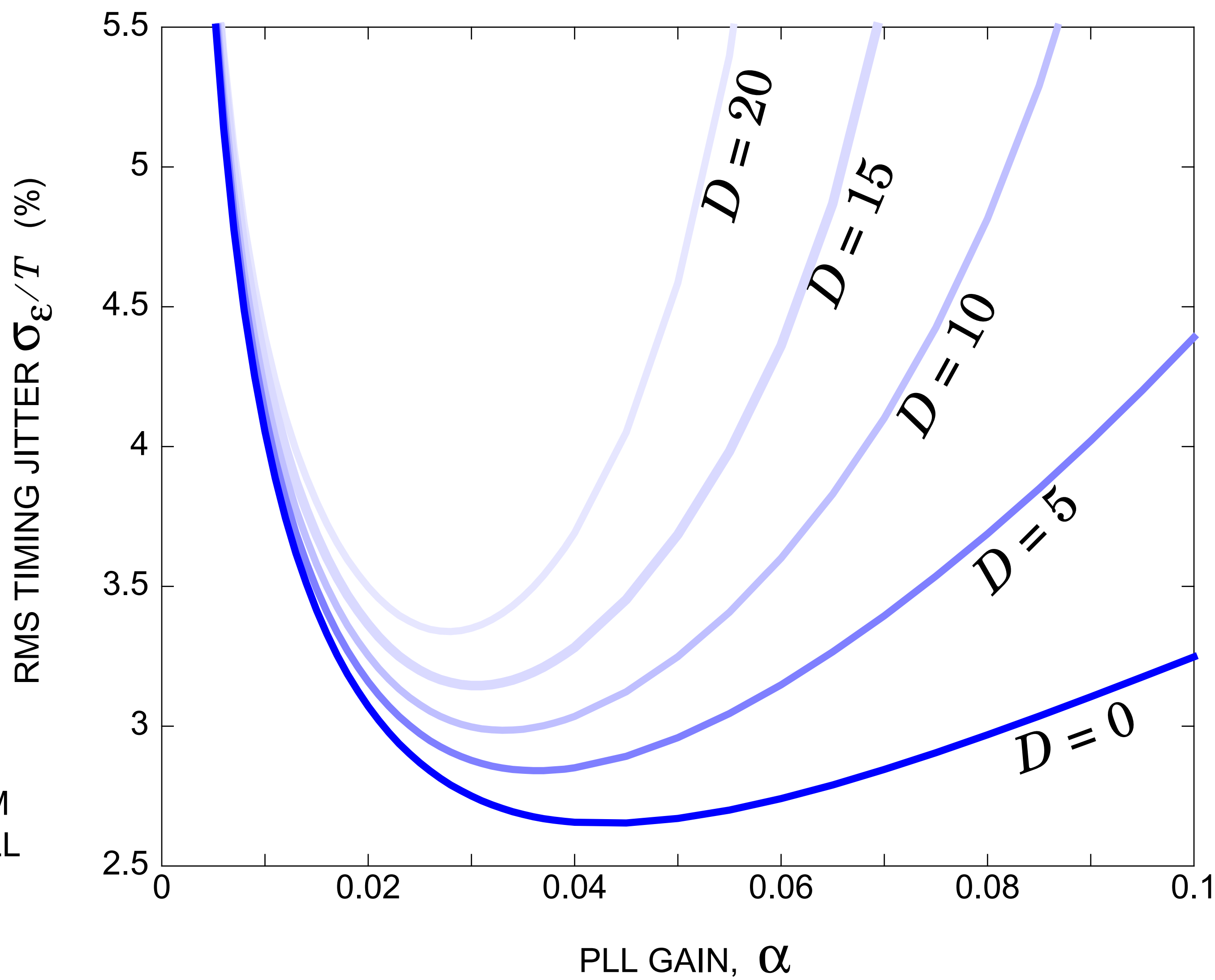
Three places to get decisions:



Inherent trade-off: reliability versus delay.

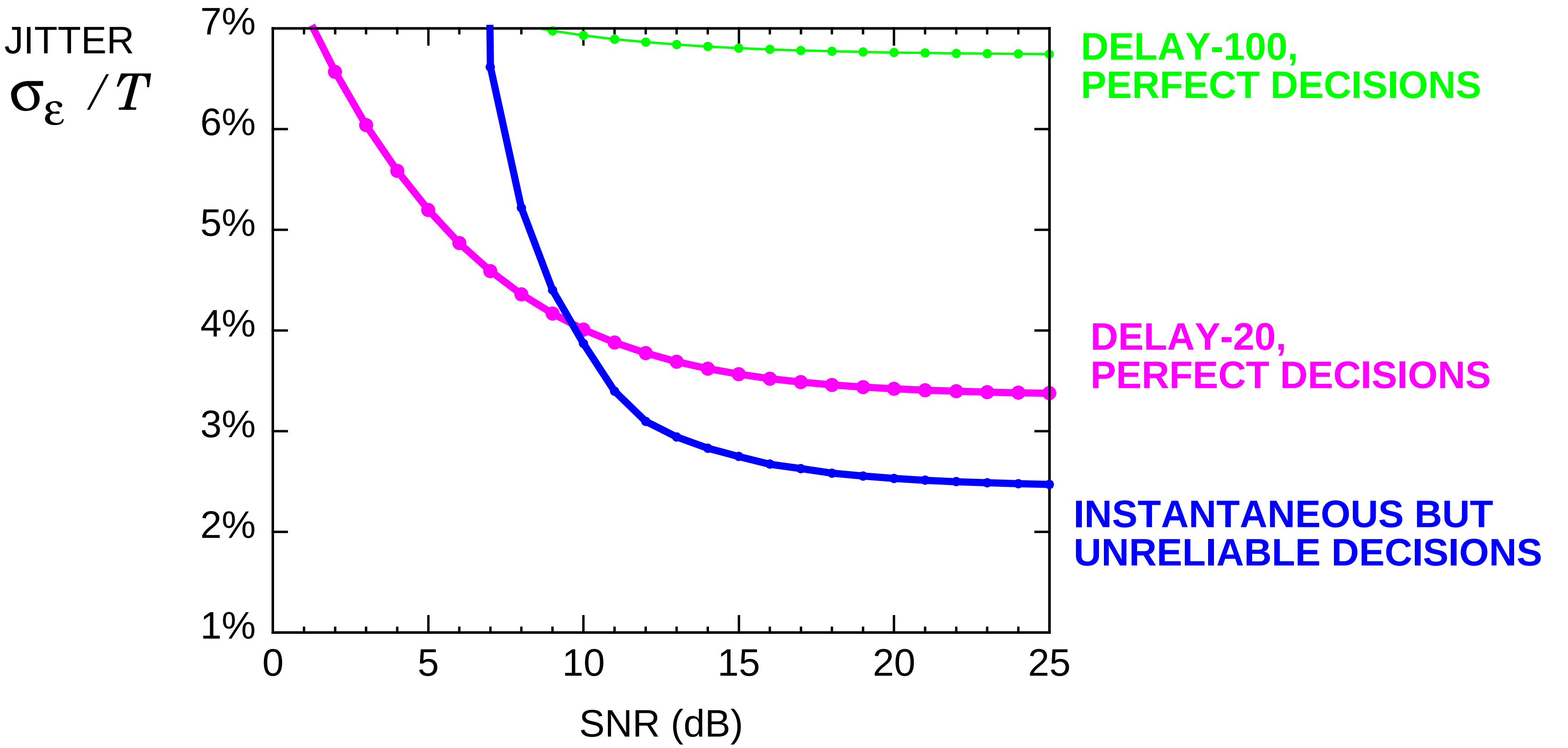
- to get more reliable decisions requires more decoding delay D
- delay decreases agility to timing variations

Decision Delay Degrades Performance



TRAINED M&M
1st-ORDER PLL
E_b / N_0 = 8 dB
σ_w / T = 0.5%

The Instantaneous-vs-Reliable Trade-Off



Parameters

Averaged over 40,000 bits
random walk σ_w / T = 0.5%
1st-order M&M PLL
α optimized for SNR = 10 dB

Delay	α_{opt}
100	**0.006**
20	**0.017**
0	**0.046**

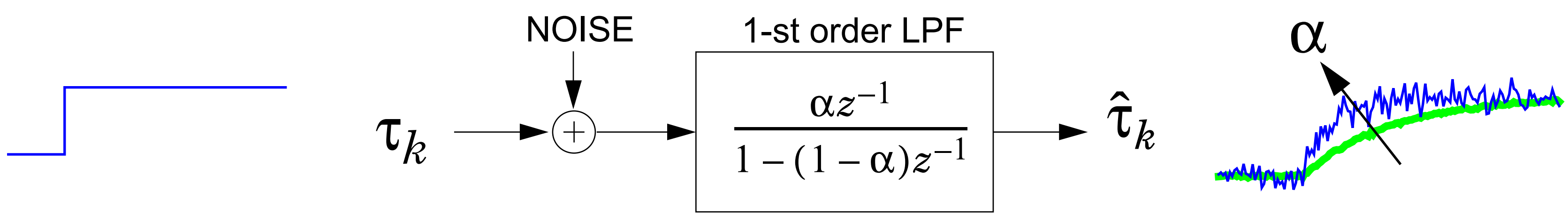
Reliability becomes more important at low SNR.

Linearized Analysis

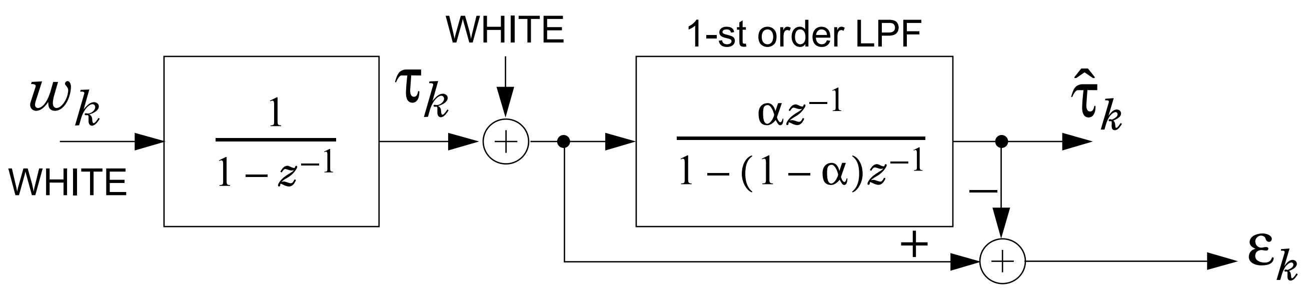
Assume $\hat{\varepsilon}_k = \varepsilon_k + \text{independent noise}$

$$= \tau_k - \hat{\tau}_k + n_k$$

$\Rightarrow$1st order PLL, $\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha(\tau_k - \hat{\tau}_k + n_k)$, is a linear system:



Ex: Random walk



$\Rightarrow$ derive optimal α to minimize σ_ε^2

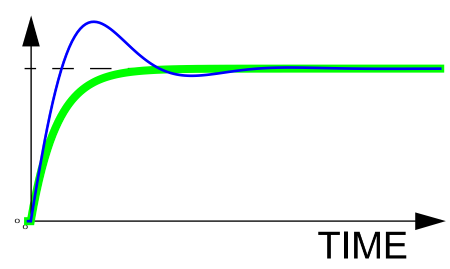
The PLL Update

1st-order PLL: $$\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha\hat{\varepsilon}_k$$

- Already introduced using LMS
- Easily motivated intuitively:
  - ➢ if $\hat{\varepsilon}_k$ is accurate, $\alpha = 1$ corrects in one step
  - ➢ Smaller α attenuates noise at cost of slower response

2nd-order PLL: $$\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha\hat{\varepsilon}_k + \beta\sum_{n=-\infty}^{k}\hat{\varepsilon}_n$$

- Accumulate TED output to anticipate trends
- P+I control
- Closed-loop system is second-order LPF
- Faster response
- Zero steady-state error for frequency offset

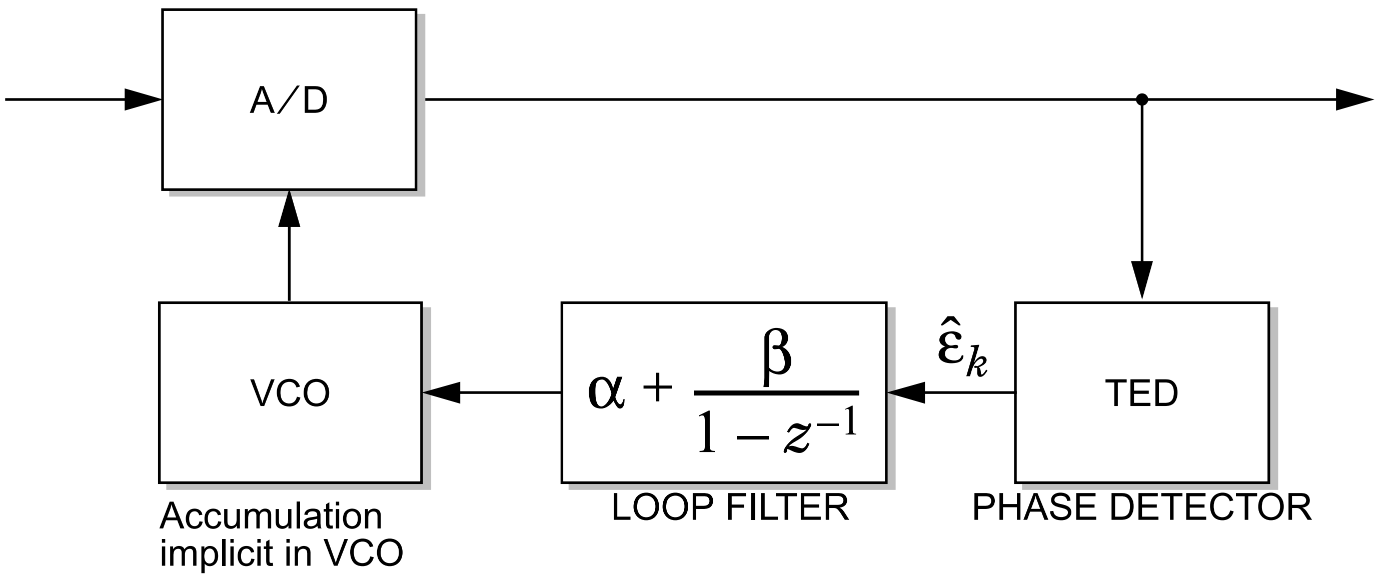


Equivalent Views of PLL

Analysis: Sample at $\{kT + \hat{\tau}_k\}$, where

- $\hat{\tau}_{k+1} = \hat{\tau}_k + \alpha\hat{\varepsilon}_k + \beta\sum_{n=-\infty}^{k} \hat{\varepsilon}_n$,
- $\hat{\varepsilon}_k$ is estimate of timing error at time k .

Implementation:



Iterative Timing Recovery

Motivation

- Powerful codes $\Rightarrow$ low SNR $\Rightarrow$ timing recovery is difficult
- traditional PLL approach ignores presence of code

Key Questions

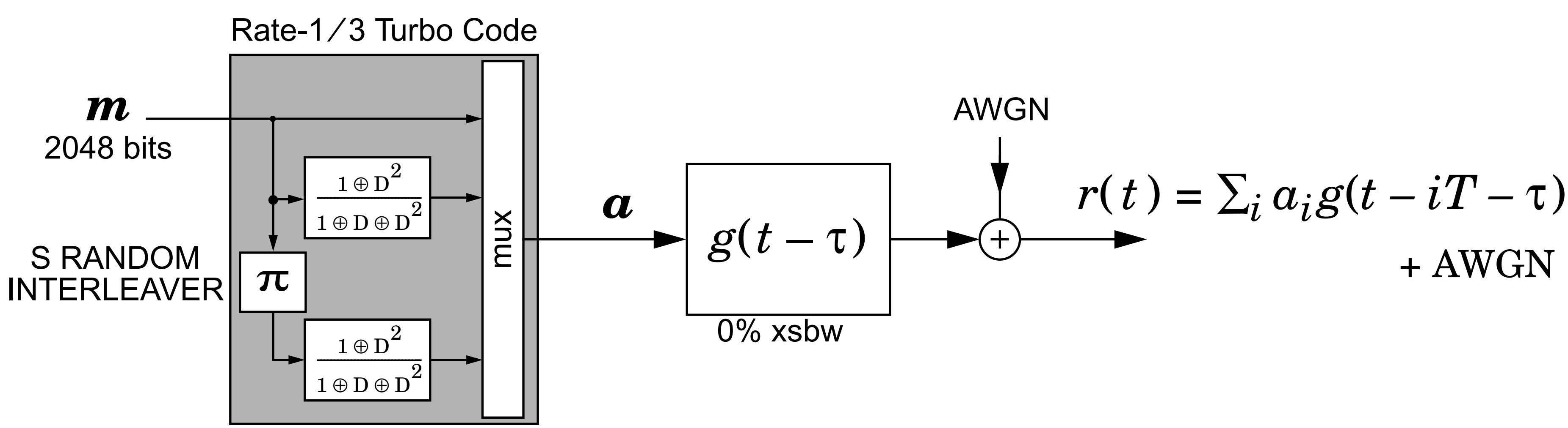
- How can timing recovery exploit code?
- What performance gains can be expected?
- Is it practical?

A Canonical Example

Simplest possible channel model:

- $\{\pm 1\}$ alphabet, ideal ISI-free pulse shape
- constant timing offset τ
- AWGN.

Add a rate-1/3 turbo code with $\{\pm 1\}$ alphabet:



Problem: Recover message in face of unknown noise, timing offset

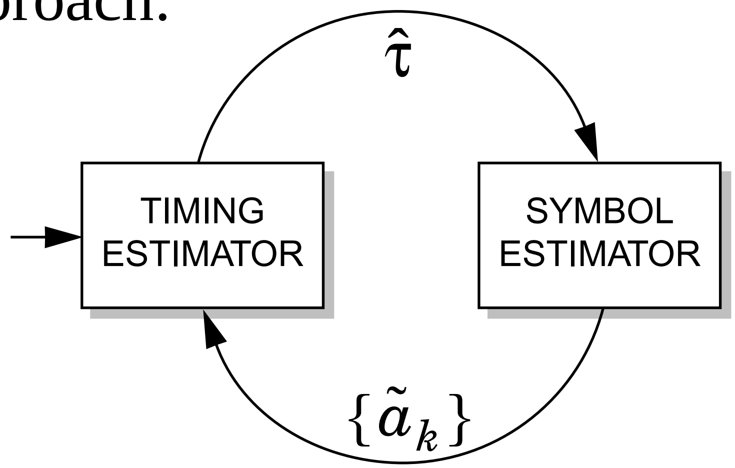
Iterative ML Timing Recovery

The ML estimator *with training* minimizes:

$$J(\tau \mid \boldsymbol{a}) = \int_{-\infty}^{\infty} \left(r(t) - \sum_i a_i g(t - iT - \tau) \right)^2 dt$$

Without training, the ML estimator minimizes $E_{\boldsymbol{a}}[J(\tau \mid \boldsymbol{a})]$.

An EM-like approach:



Useful in concept, but overstates complexity.

For example, the timing estimator might itself be iterative:

$$\hat{\tau}_{i+1} = \hat{\tau}_i - \mu J'(\hat{\tau}_i \mid \{\tilde{a}_i\}) \ .$$

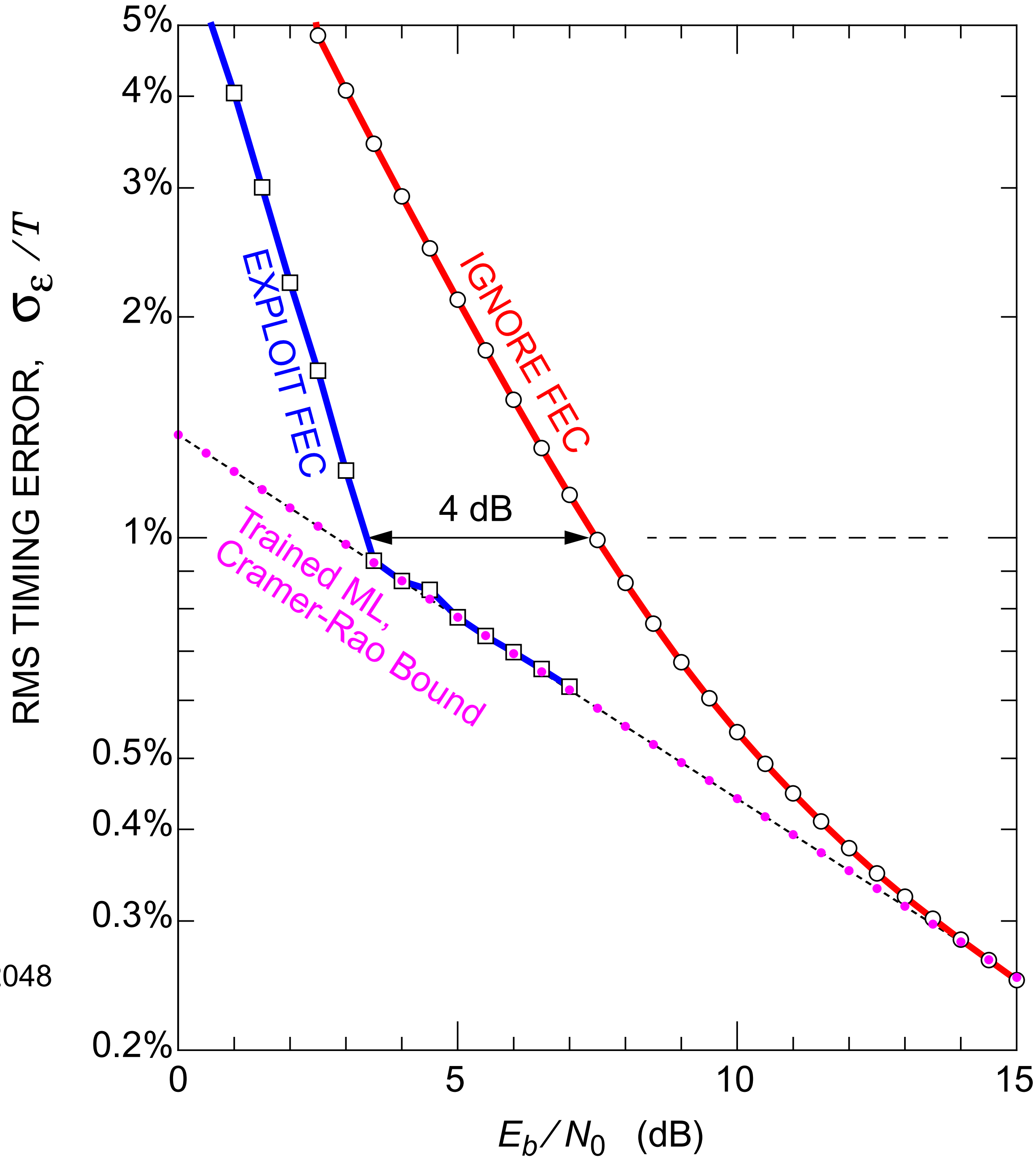
A Reduced-Complexity Approach

Collapse three loops to a single loop.

```
Initialize τ̂_0
Iterate for i = 0, 1, 2, …
   decode component 1
   decode component 2
   update timing estimate, τ̂_{i+1} = τ̂_i − μJ′(τ̂_i | {ã_i})
   interpolate
end
```

As a benchmark, an iterative receiver that *ignores* the presence of FEC will replace the pair of decoders by $\tilde{a}_k^{(i)} = \tanh((r(kT + \hat{\tau}_i))/\sigma^2)$.

Results



Parameters

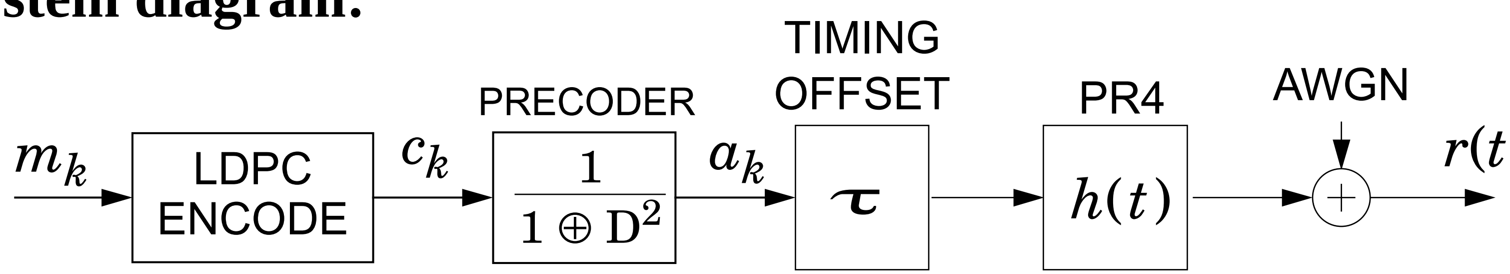
τ = 0.123T, AWGN channel
Rate-1/3 Turbo Code
K = 2048 message bits
N = 6150 coded bits
S = 16 random interleaver, length 2048
1 inner per outer iteration
Averaged over 180 trials
$\sigma_\varepsilon^2 = E[(\hat{\tau} - \tau)^2]$

New Model: Random Walk and ISI

Equivalent equalized readback waveform:

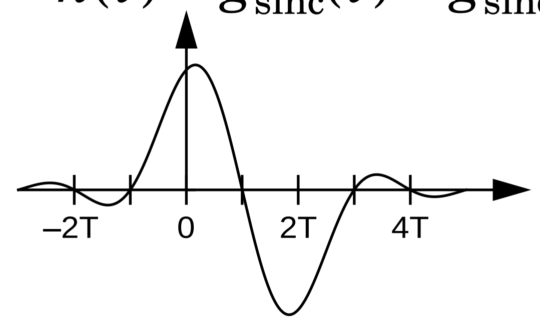
$$r(t) = \sum_k a_k h(t - kT - \tau_k) + \text{AWGN} .$$

System diagram:



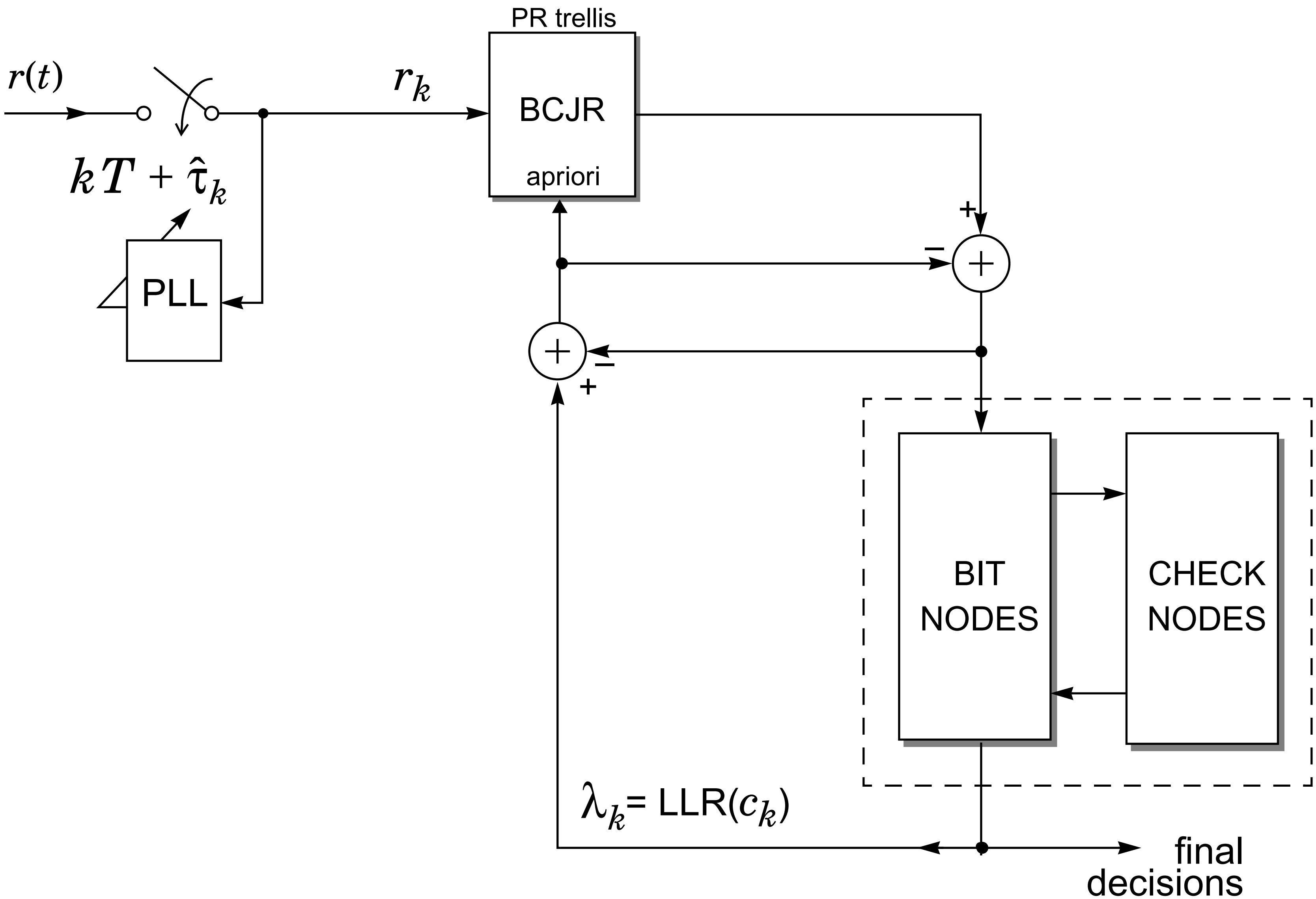
Rate-8/9
(4095, 3640)
irregular LDPC
code with node-degree
distribution polynomials:

$$h(t) = g_{\text{sinc}}(t) - g_{\text{sinc}}(t - 2T)$$

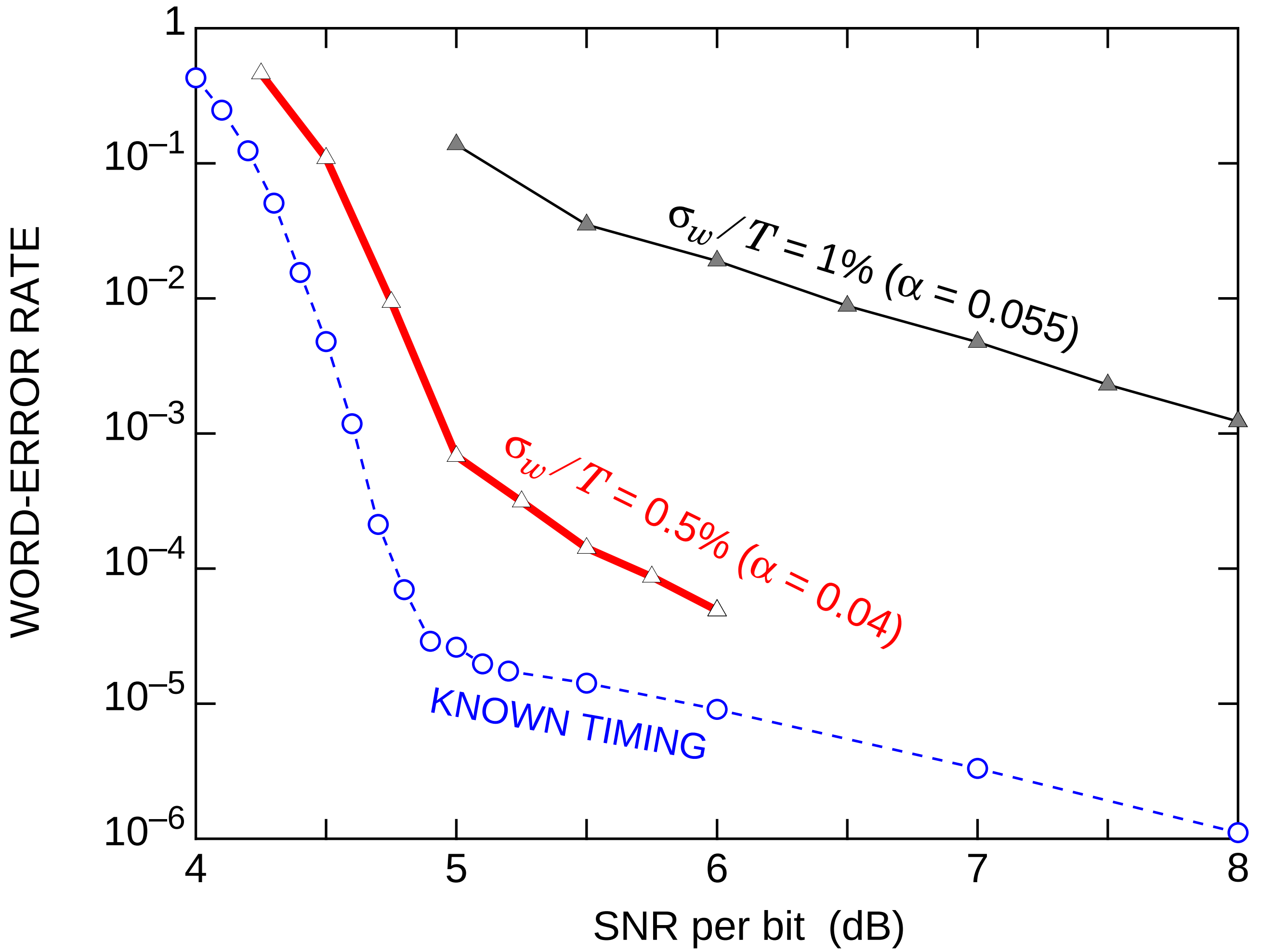


$$\lambda_{\text{bit}}(x) = 0.38767x^2 + 0.39823x^3 + 0.14688x^6 + 0.06722x^7$$
$$\rho_{\text{check}}(x) = 0.10309x^{29} + 0.89691x^{30}$$

Conventional Turbo Equalizer + PLL



Performance of Conventional Approach

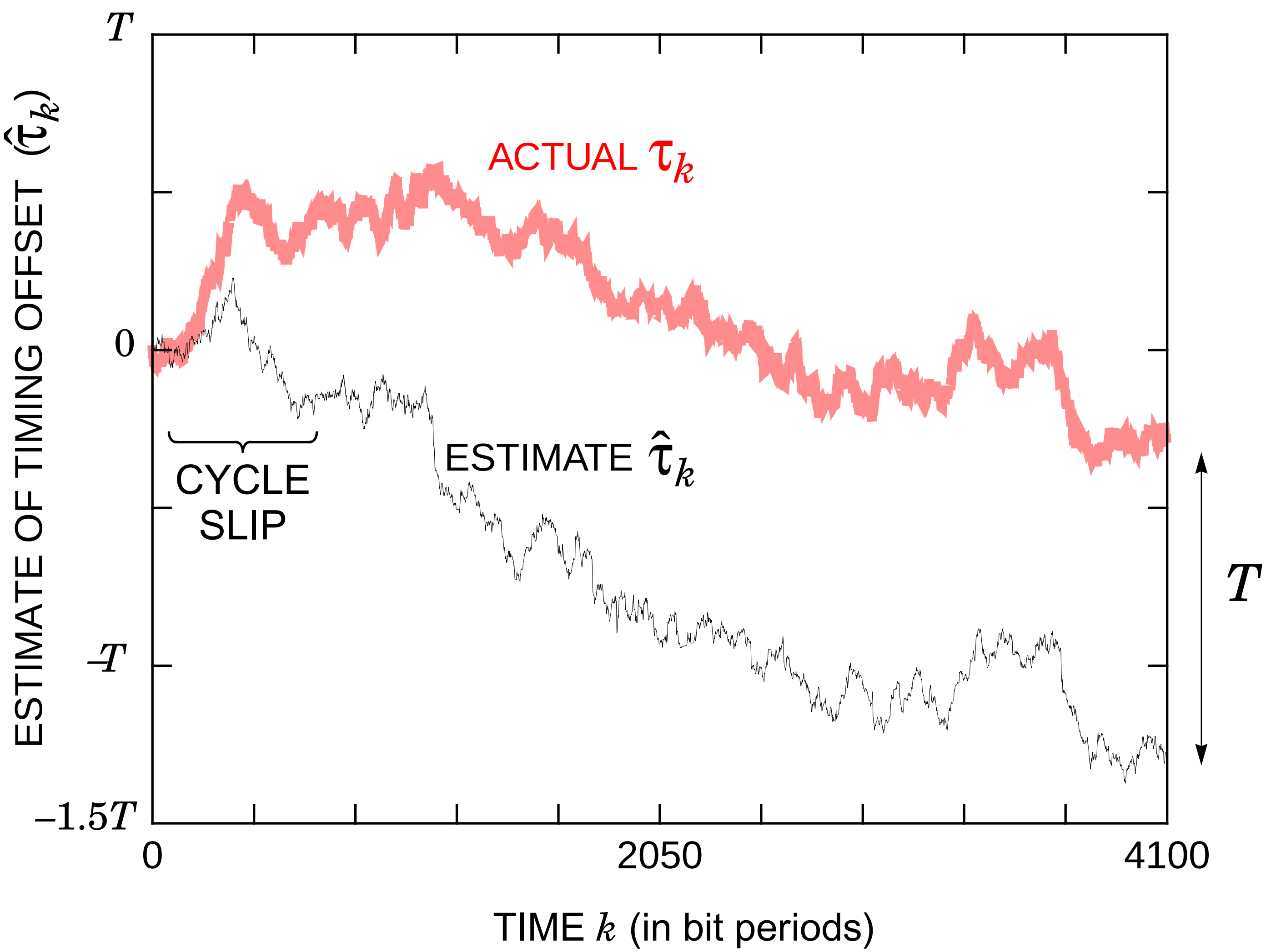


Parameters

Known Timing: 10 ⁄ 5
Conventional: 25 ⁄ 5
max 10000000 words

Big penalty as σ_w / T increases: cycle slips.

Cycle-Slip Example

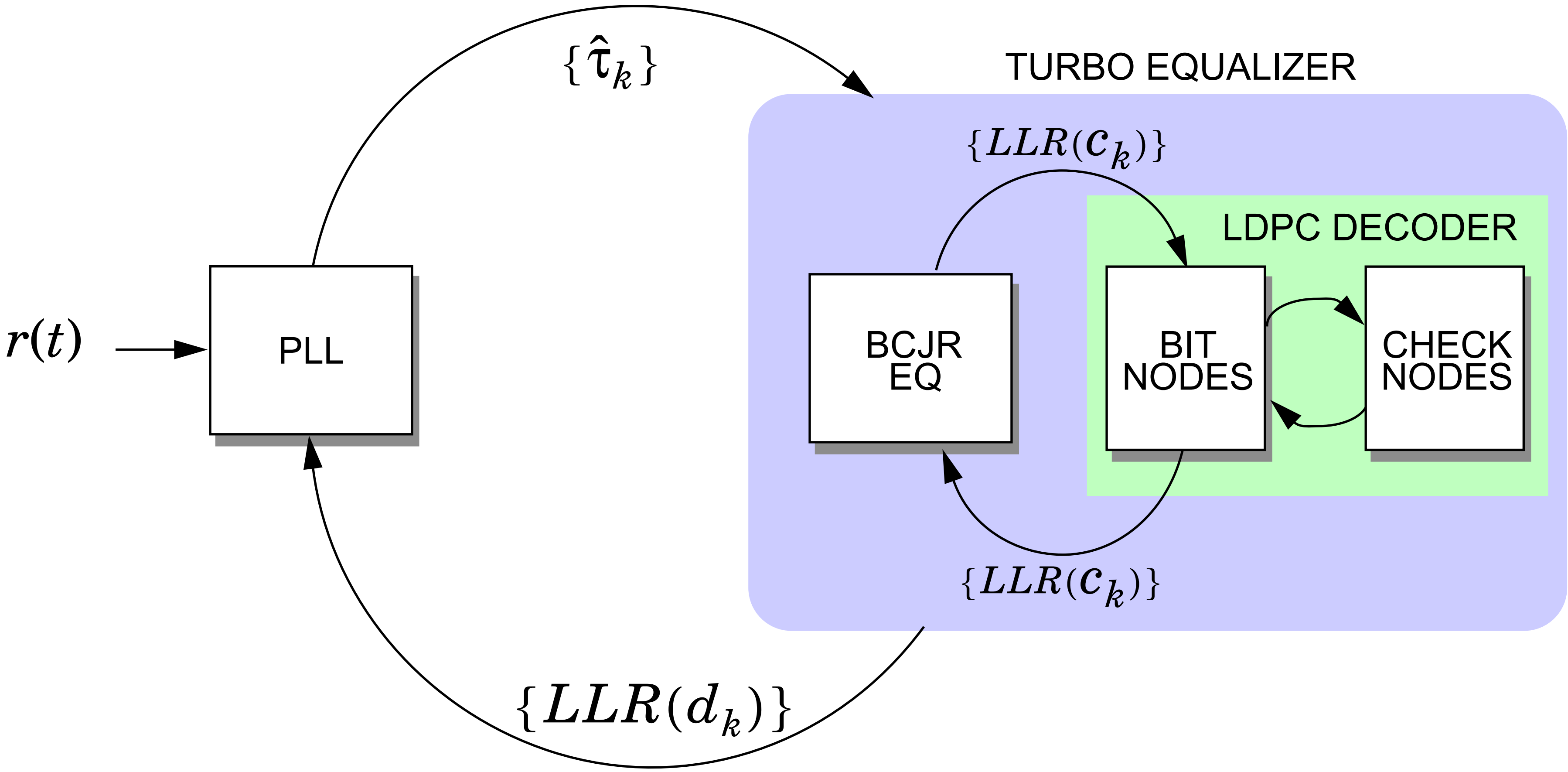


Parameters

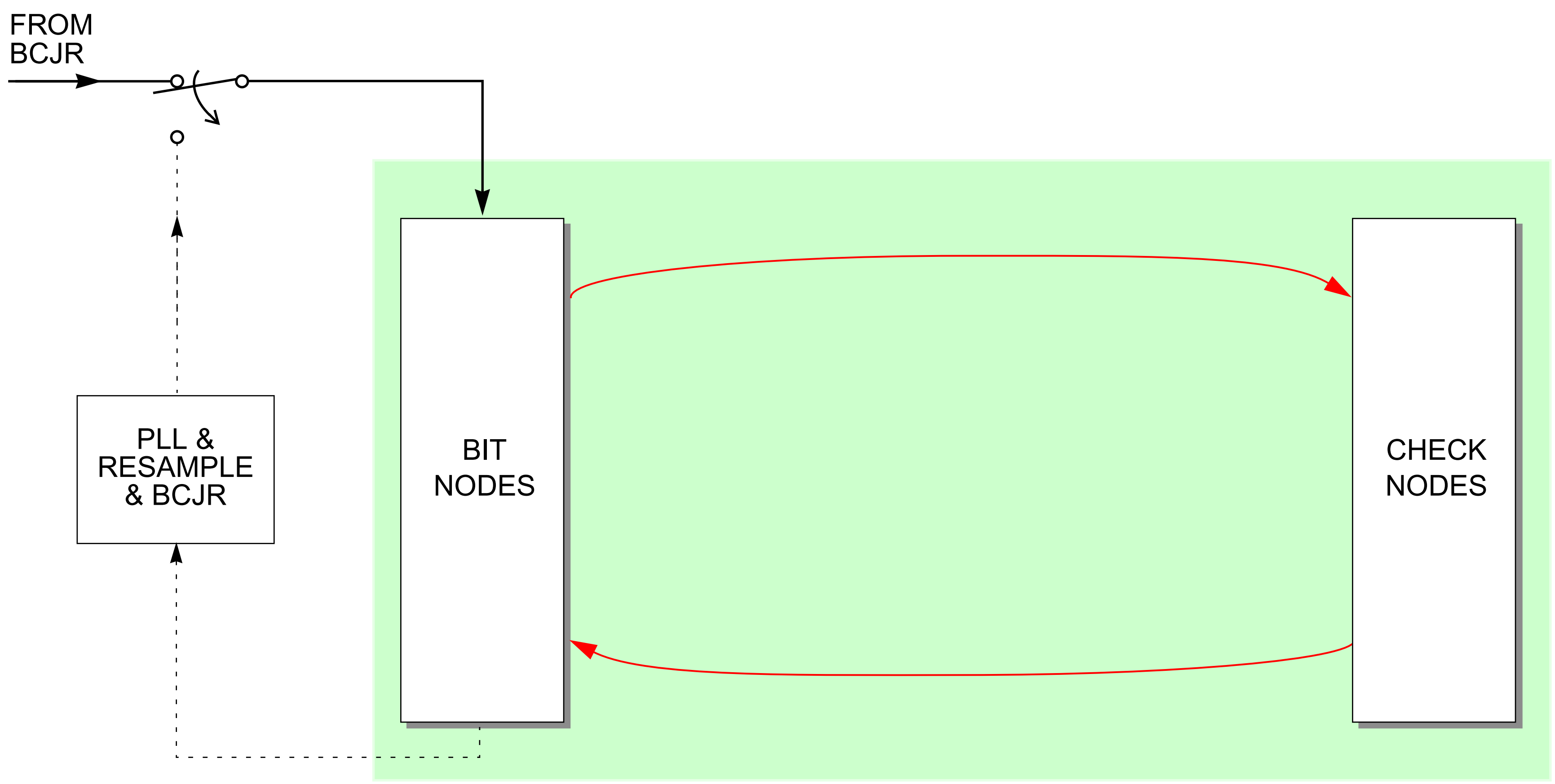
SNR_{bit} = 5.0 dB
σ_w / T = 1.0%
α = 0.055

Nested Loops

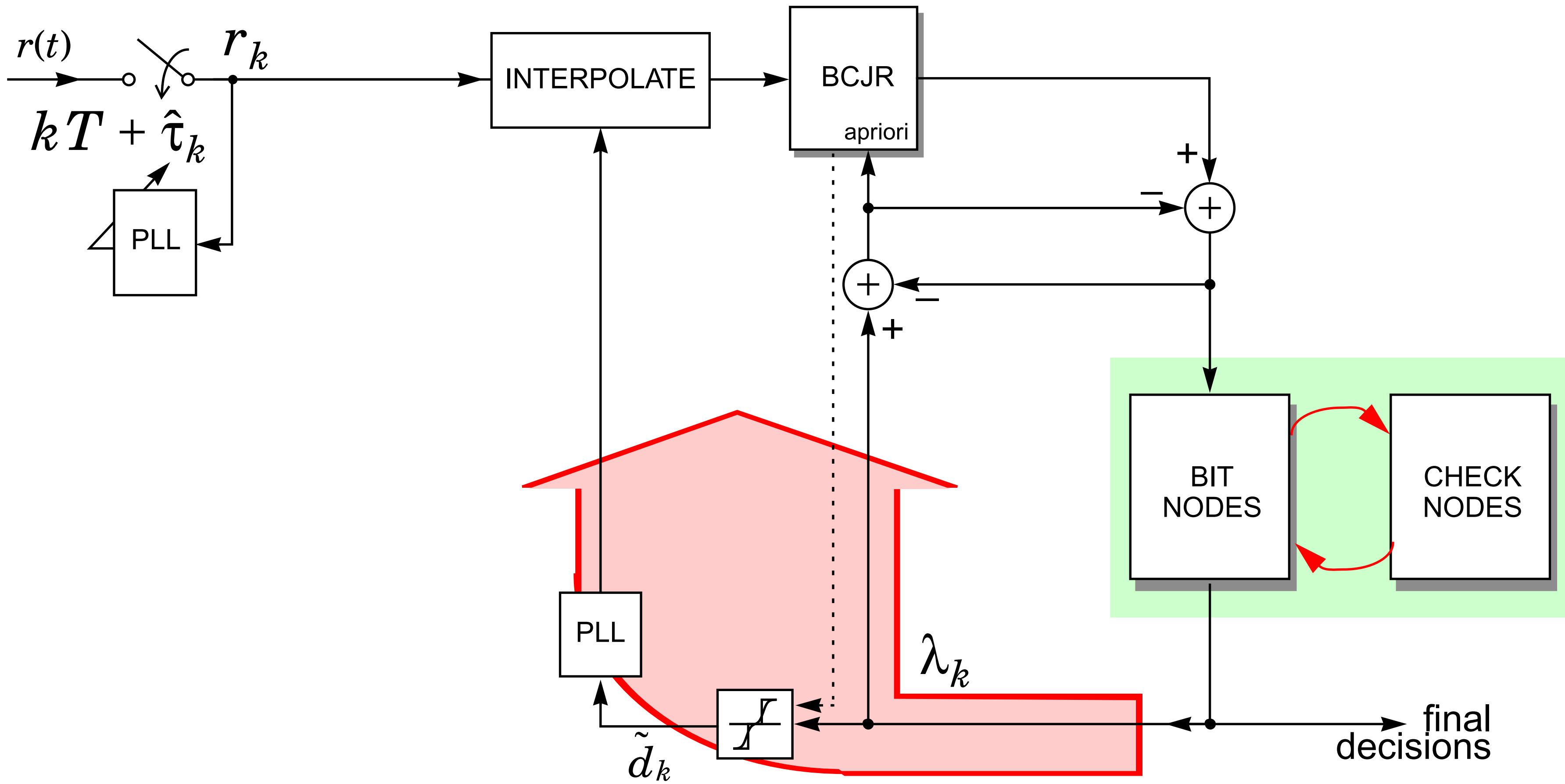
Iterate between PLL and turbo equalizer, which in turn iterates between BCJR and LDPC decoder, which in turn iterates between bit nodes and check nodes.



A Decoder-Centric View

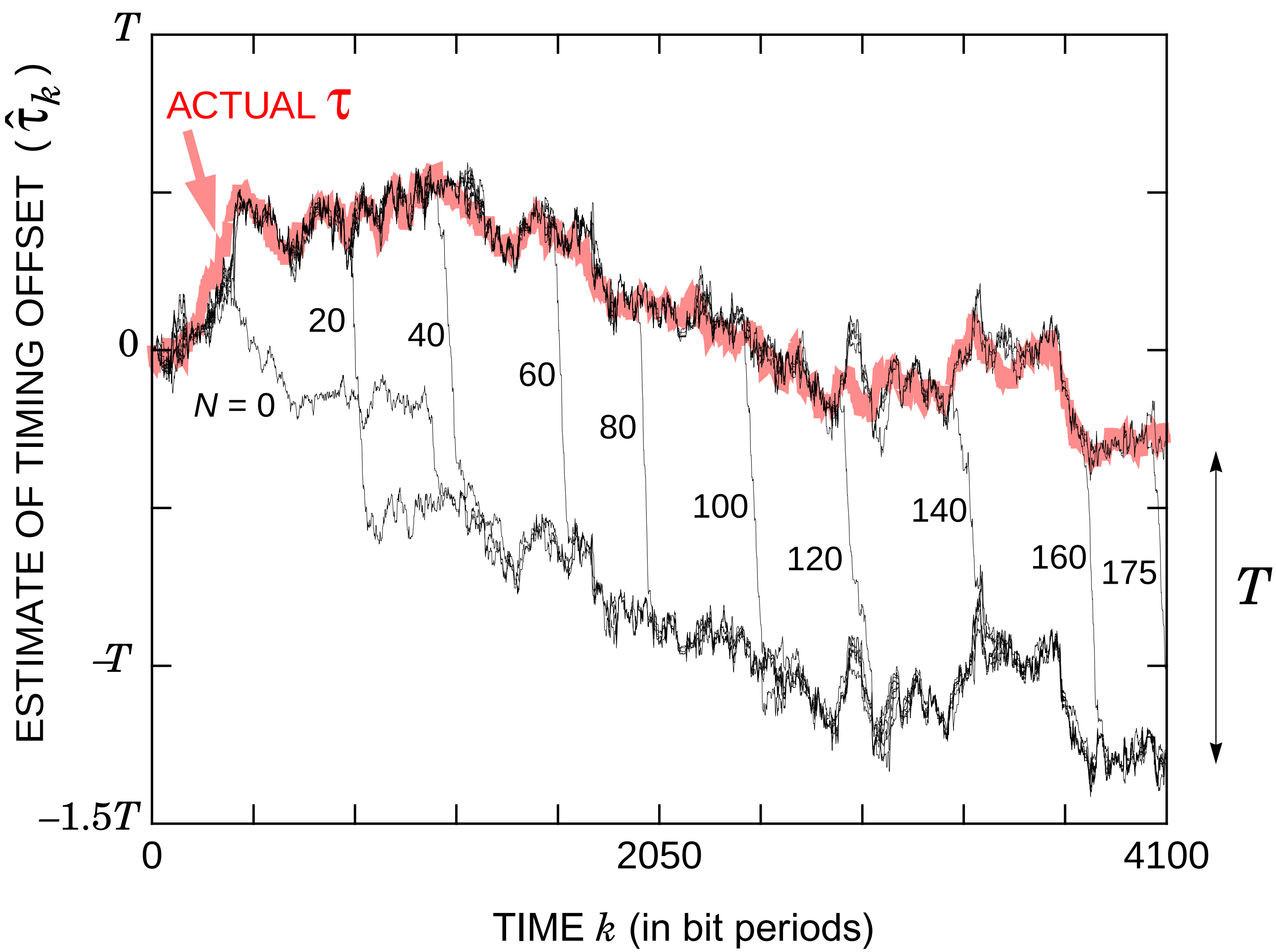


Iterative Timing Recovery and TEQ



Automatic Cycle-Slip Correction

Iterative receiver automatically corrects for cycle slips:

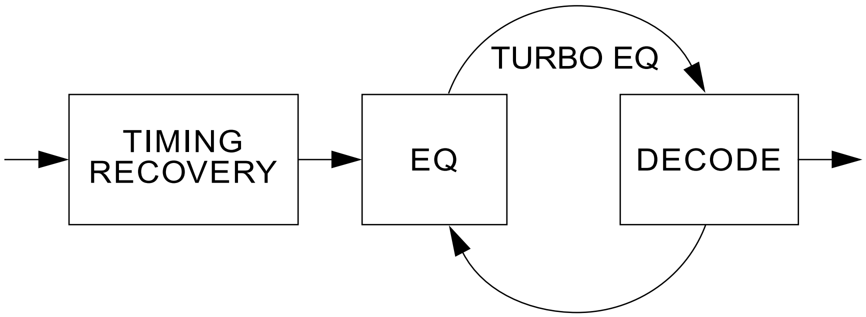


Parameters

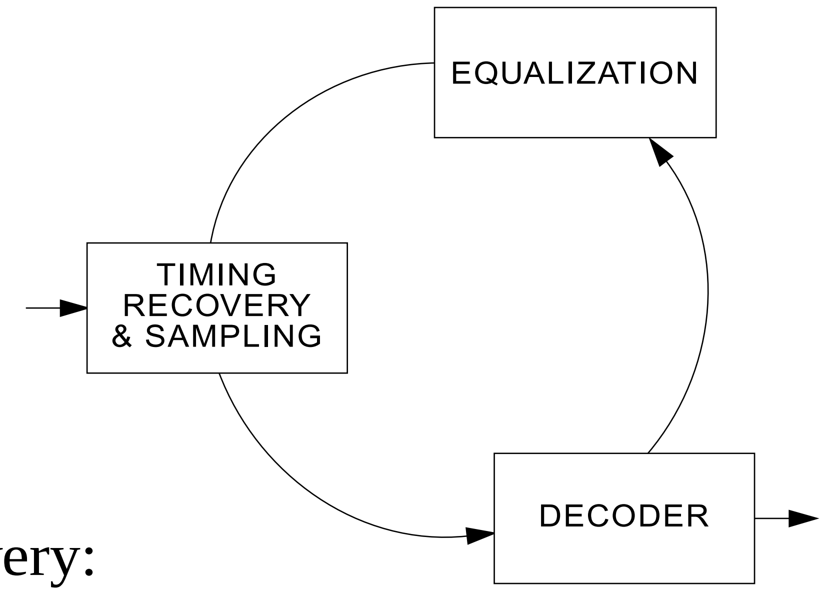
SNR_{bit} = 5.0 dB
σ_w / T = 1.0%
α = 0.055
N/5 iterations

3 Approaches to Timing Recovery

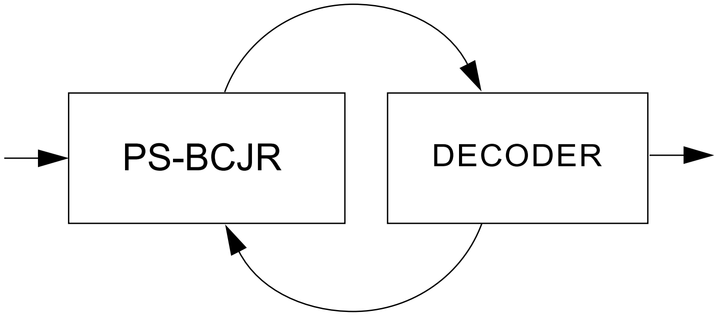
Conventional:



3-Way
Iterative Timing Recovery:



Per-Survivor Iterative Timing Recovery:

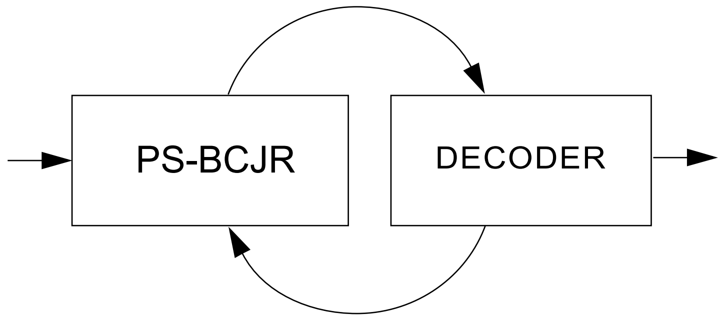


Per-Survivor Processing [Raheli, Polydoros, Tzou 91-95]

- A general framework for estimating Markov process with unknown parameters and independent noise
- Basic idea: Add a separate estimator to each survivor of Viterbi algorithm
- Has been applied to channel identification, adaptive sequence detection, carrier recovery
- Application to timing recovery [Kovintaveat *et al.*, ISCAS 2003]:
  - ❑ Start with traditional Viterbi algorithm on PR trellis
  - ❑ Run a separate PLL on each survivor, based on its decision history
  - ❑ Motivations:
    - ➊ PLL is *fully trained* whenever correct path is chosen!
    - ➋ Can avoid decision delay altogether

Per-Survivor BCJR?

Motivation: Exploit PSP concept in iterative receiver:



Problem: BCJR algorithm has no "survivors".

Proposal: Add depth-one "survivor" for purposes of timing recovery only.

The result is *PS-BCJR*

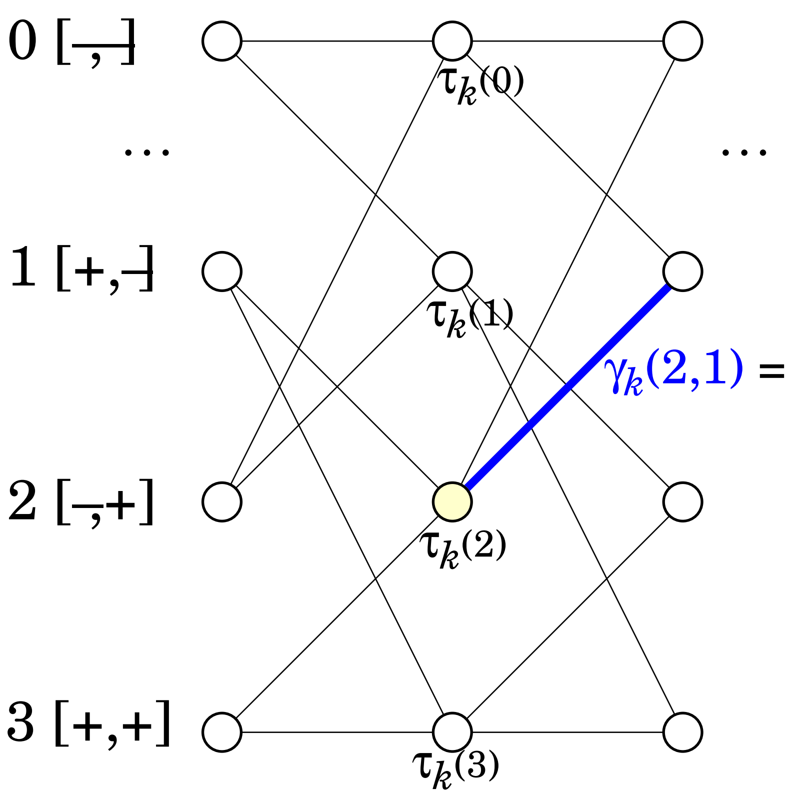
- ❑ Start with traditional BCJR algorithm on PR trellis
- ❑ Embed timing-recovery process inside
- ❑ Run multiple PLL's in parallel, one for each "survivor"

PS-BCJR Branch Metric

Key: Each node $p \in \{0,1,2,3\}$ in trellis at time k has *its own*

- $\tau_k(p)$, an estimate of the timing offset τ_k.
- $r_k(p) = r_k(kT + \tau_k(p))$, corresponding sample

The branch metrics depend on samples of the starting state:

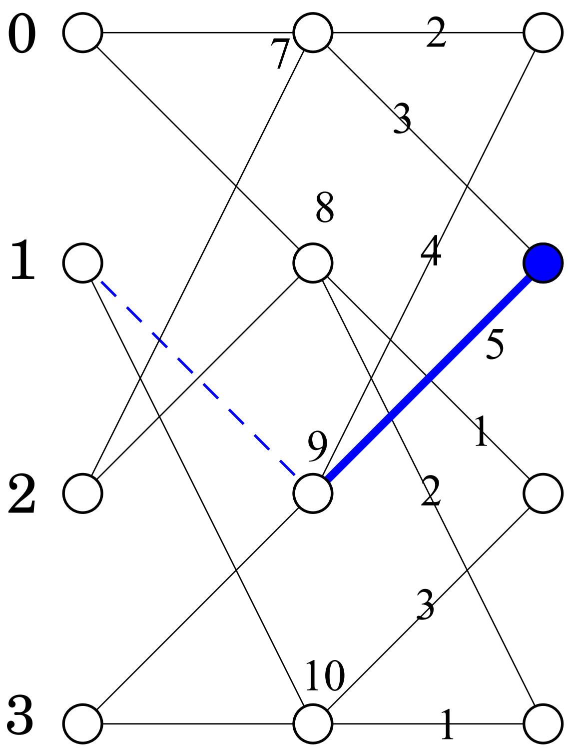


$$\gamma_k(2,1) = \exp\{-|\, r_k(2) - d^{(2,1)} |^2 / (2\sigma^2) + a^{(2,1)} \lambda_k^a / 2\}$$

Per-Survivor BCJR: Forward Recursion

Associate with each node $p \in \{0, 1, 2, 3\}$ at time k the following:

- forward metric $\alpha_k(p)$
- *predecessor* $\pi_k(p)$
- *forward timing offset estimate* $\tau_k(p)$



$$\begin{cases} \alpha_{k+1}(1) = 7 \times 3 + 9 \times 5 = 66 \\ \pi_{k+1}(1) = \text{argmax}_{0,2}\{7 \times 3,\ 9 \times 5\} = 2 \\ \tau_{k+1}(1) = \tau_k(2) + \mu(\underbrace{r_k(2)}_{r(kT + \tau_k(2))} d^{(\pi_k(2),2)} - \underbrace{r_{k-1}(\pi_k(2))}_{r((k-1)T + \tau_{k-1}(\pi_k(2)))} d^{(2,1)}) \end{cases}$$

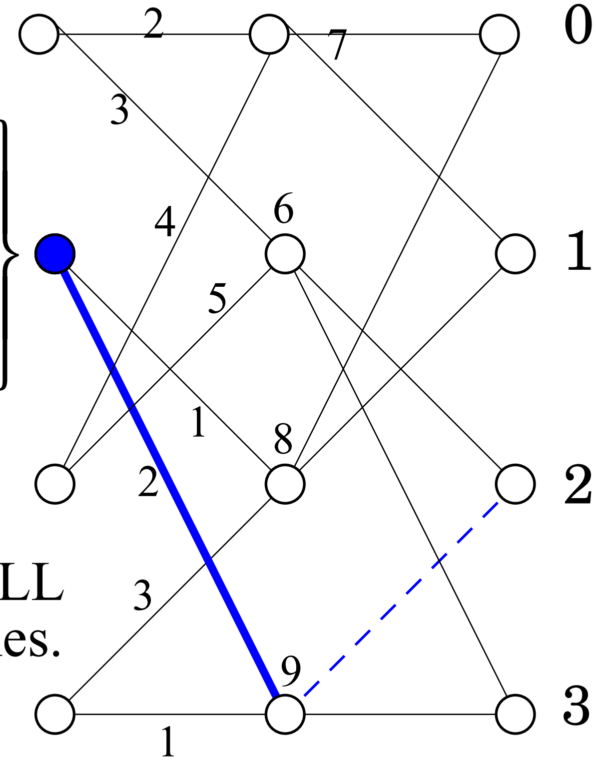
Notation complicated, but idea is simple:
update blue node timing using M&M PLL driven by the samples & inputs corresponding to blue branches.

Backward Recursion

Associate with each node $p \in \{0,1,2,3\}$ at time k the following

- backward metric $\beta_k(p)$
- *successor* $\sigma_k(p)$
- *backward timing offset estimate* $\tau_k^b(p)$

$$\left\{\begin{array}{l} \beta_k(1) = 8 \times 1 + 9 \times 2 = 26 \\ \sigma_k(1) = \text{argmax}_{2,3}\{8 \times 1, 9 \times 2\} = 3 \\ \tau_k^b(1) = \tau_{k+1}^b(3) + \mu(\underbrace{r_{k+1}(\sigma_{k+1}(3))}_{r((k+1)T + \tau_{k+2}^b(\sigma_{k+1}(3)))}d^{(1,3)} - \underbrace{r_k(3)}_{r(kT + \tau_{k+1}^b(3))}d^{(3,\sigma_{k+1}(3))}) \end{array}\right\}$$



Again, update blue node timing using *backward* M&M PLL driven by samples & inputs corresponding to blue branches.

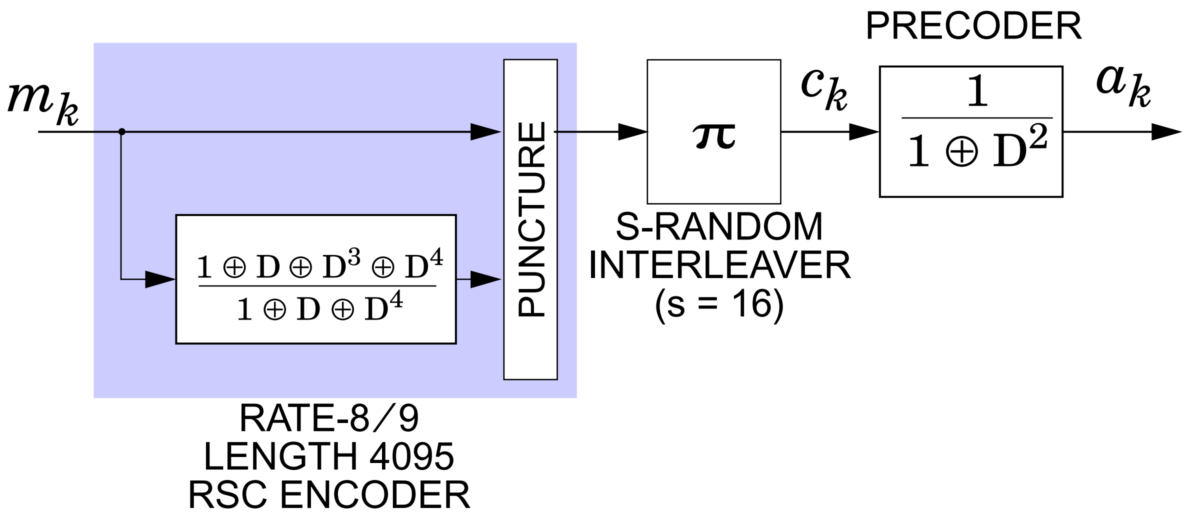
Compare Forward/Backward Timing

Backward timing estimates can exploit knowledge of forward estimates.

Option 1: Ignore forward estimates during backward pass.

Option 2: Average backward with forward estimate whenever they differ by more than some threshold (say $0.1T$) in absolute value.

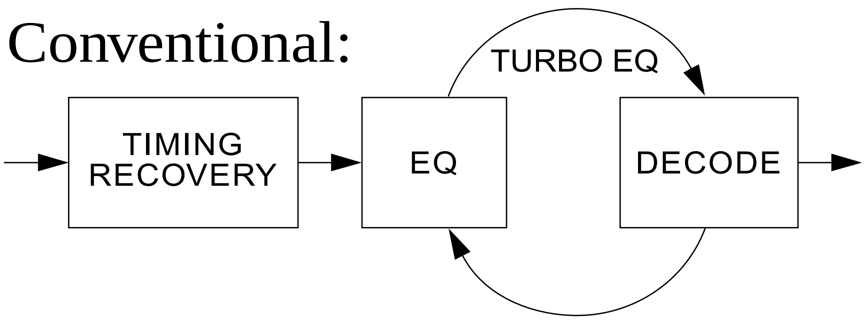
New Encoder



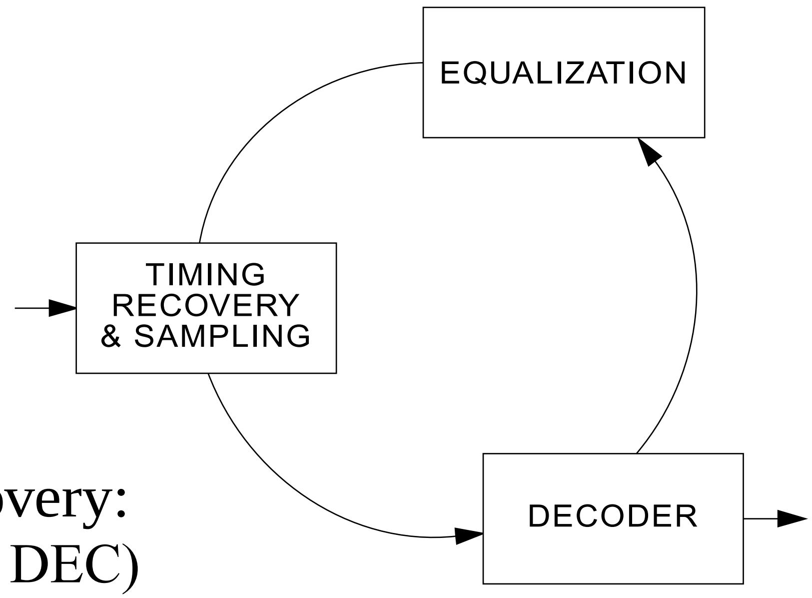
PRECODER
m_k
c_k
a_k
$\frac{1}{1 \oplus D^2}$
π
PUNCTURE
$\frac{1 \oplus D \oplus D^3 \oplus D^4}{1 \oplus D \oplus D^4}$
S-RANDOM
INTERLEAVER
(s = 16)
RATE-8/9
LENGTH 4095
RSC ENCODER

Compare 3 Systems

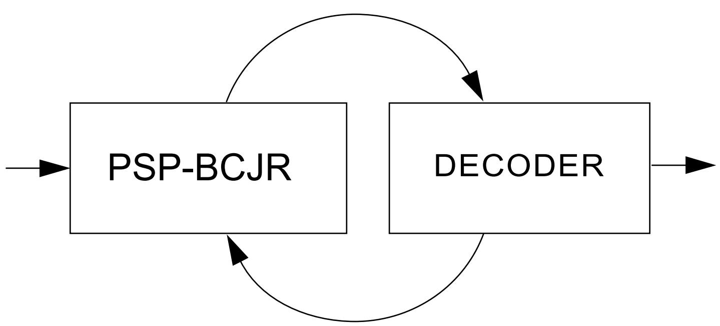
Conventional:



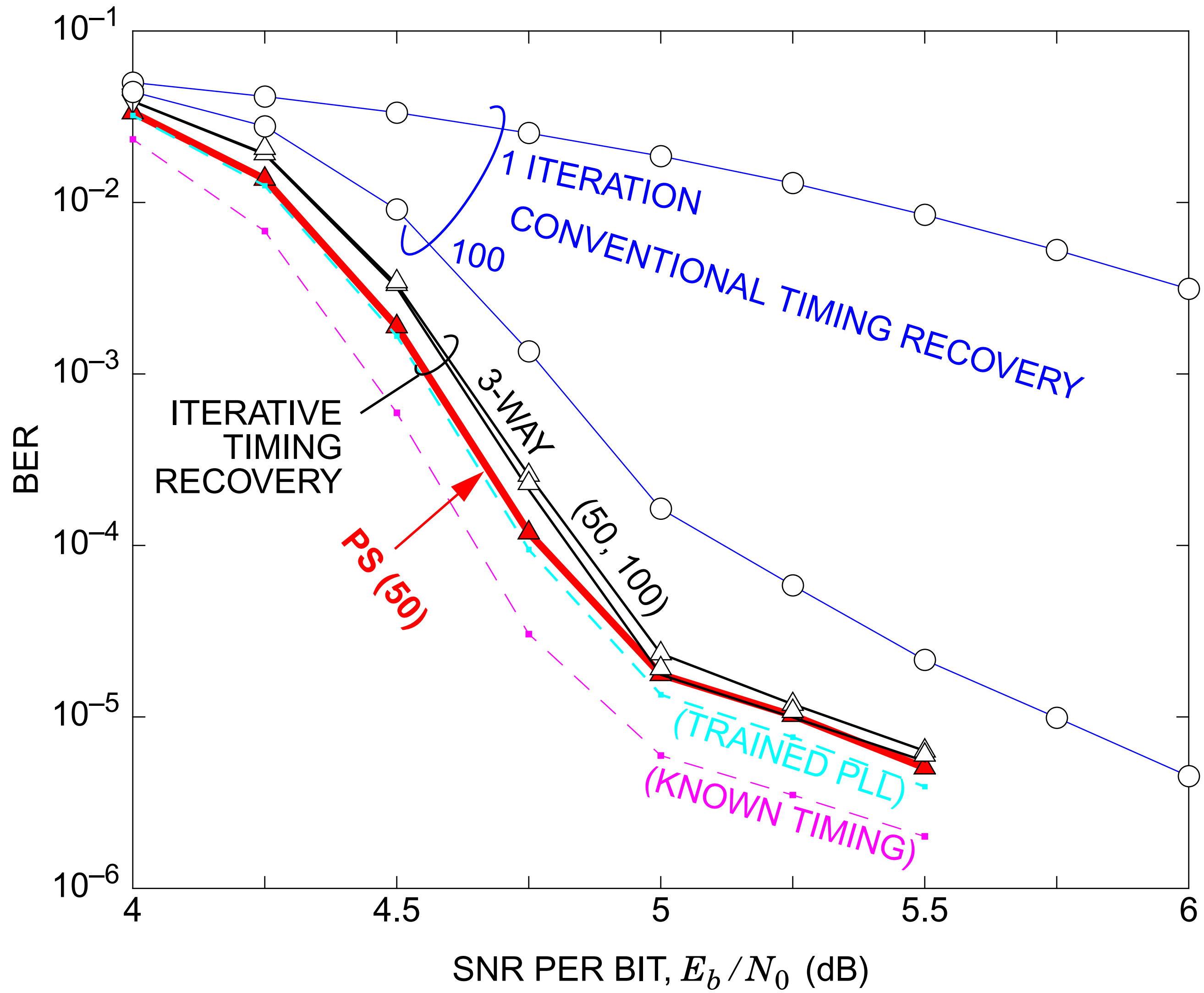
3-Way Iterative Timing Recovery:
Complexity = #IT × (PLL + BCJR + DEC)



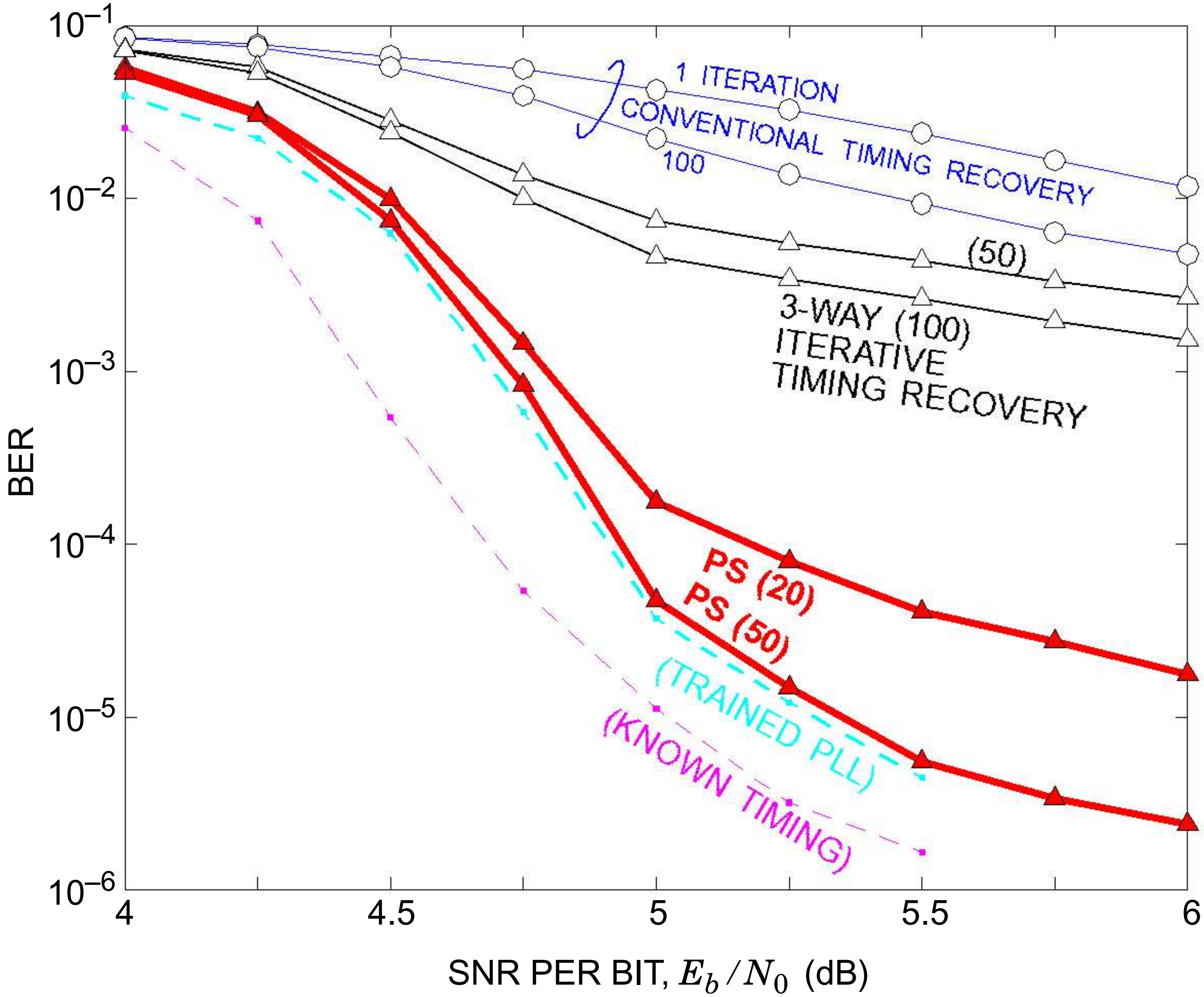
Per-Survivor Iterative Timing Recovery:
Complexity = #IT × (8 × PLL + BCJR + DEC)



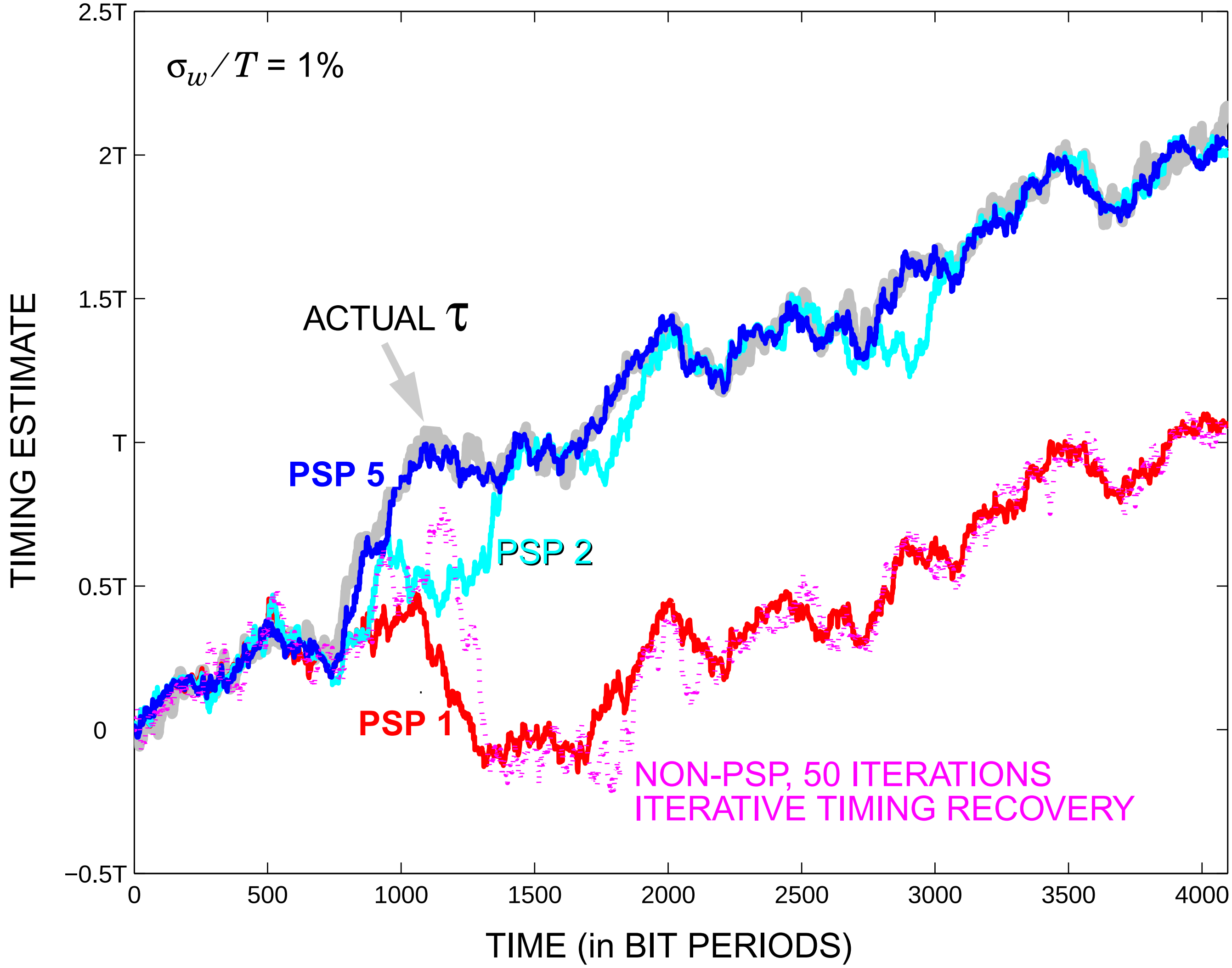
Moderate Random Walk ($\sigma_w/T = 0.5\%$)



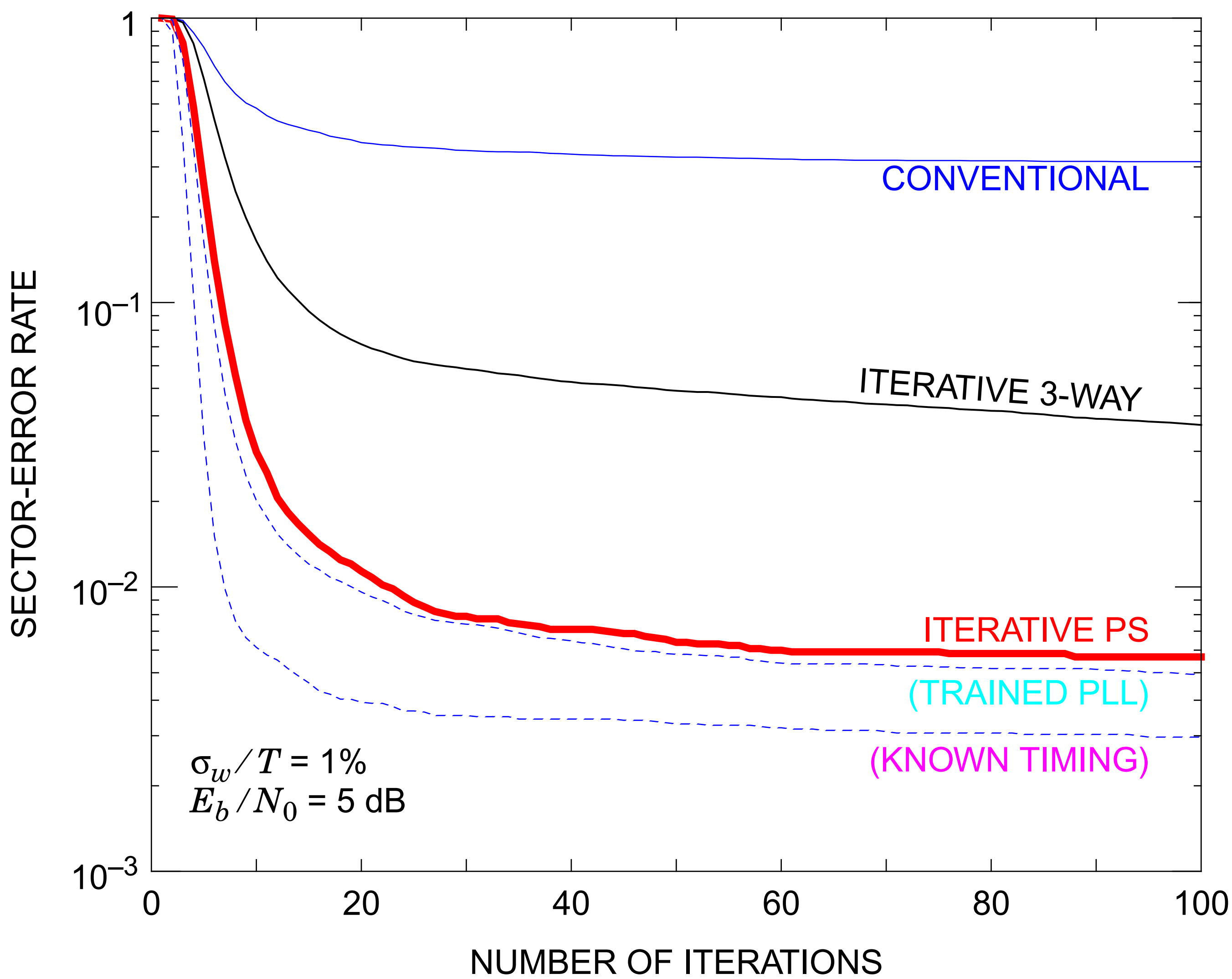
Severe Random Walk ($\sigma_w/T = 1\%$)



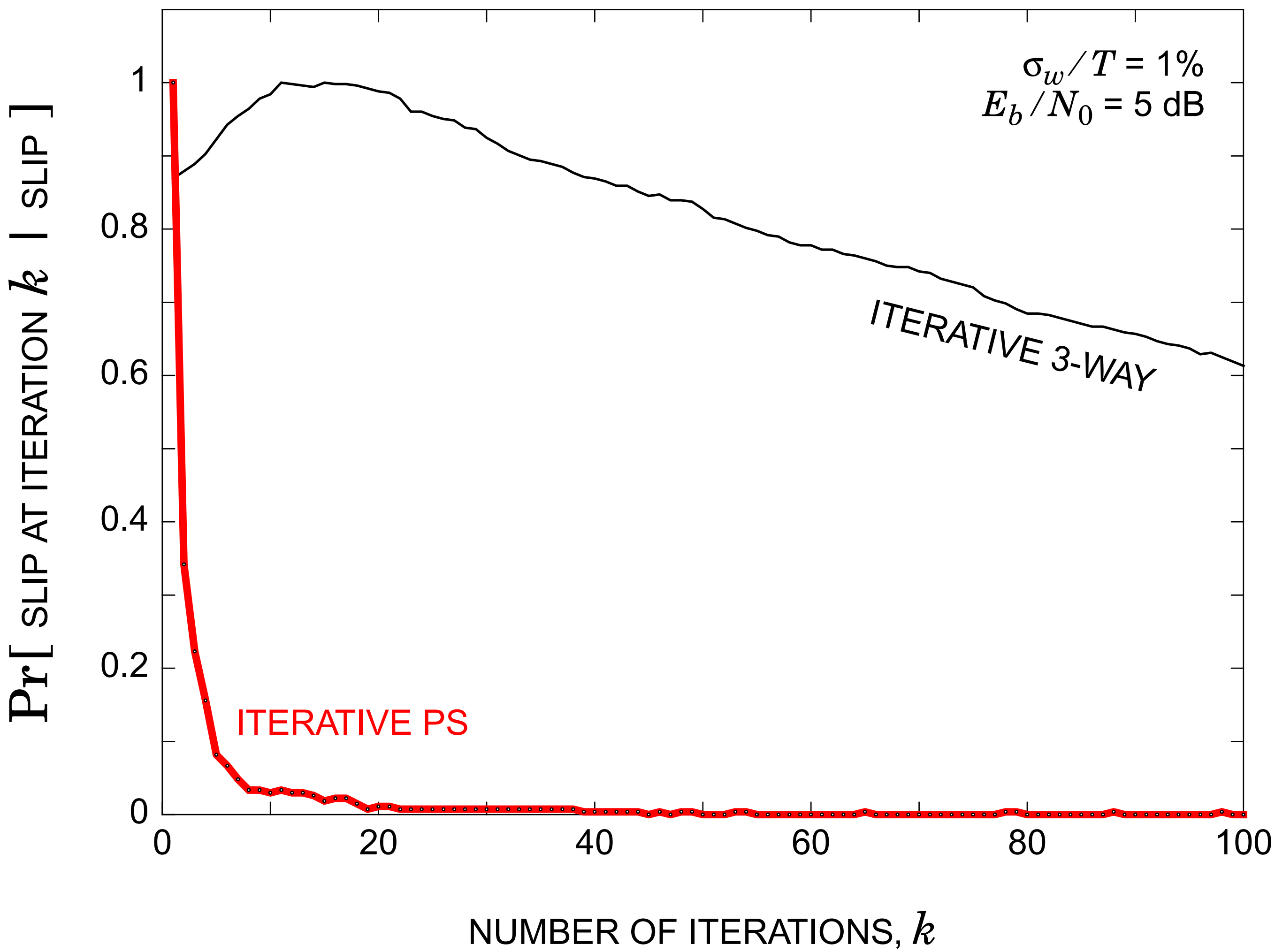
Example: PSP Corrects Quickly



Convergence Rate: ($\sigma_w / T = 1\%$)



How Long do Cycle Slips Persist?



Summary

- Powerful codes permit low SNR
  - conventional strategies fail
  - exploiting code is critical

- Problem is solvable

- We described two strategies for iterative timing recovery
  - Embed timing recovery inside turbo equalizer
  - Automatically corrects for cycle slips

- Challenges remaining
  - complexity
  - close gap to known timing