

# Viterbi's Algorithm & Trace Back

from 50<sup>th</sup> Anniversary Commemoration Talk

Keith Kumm, COMSOC Member

# Why Commemorate It?

This Could be the Digital Radio Engineer's  
1915 Moment!

So What's There to Say (that hasn't been said\*)?

- Where did this epiphany come from?
- Why is Trace Back the very essence of the algorithm?
- Beyond the algorithm, Viterbi almost predicts turbo-codes

\* in a lot of places

# Why Was It Important?

In 1966, Convolutional Codes Were Known,  
and Known to be Superior to Block Codes

**But ... We Were Missing Out on  
Their Full Performance Potential,  
And We Knew It!**

Along Came Viterbi at UCLA

By Mid 1967, We Knew How to Get at  
That Performance!

# Game Plan

There are Two Fairly Independent Parts  
of this 80+ Viewgraph Talk

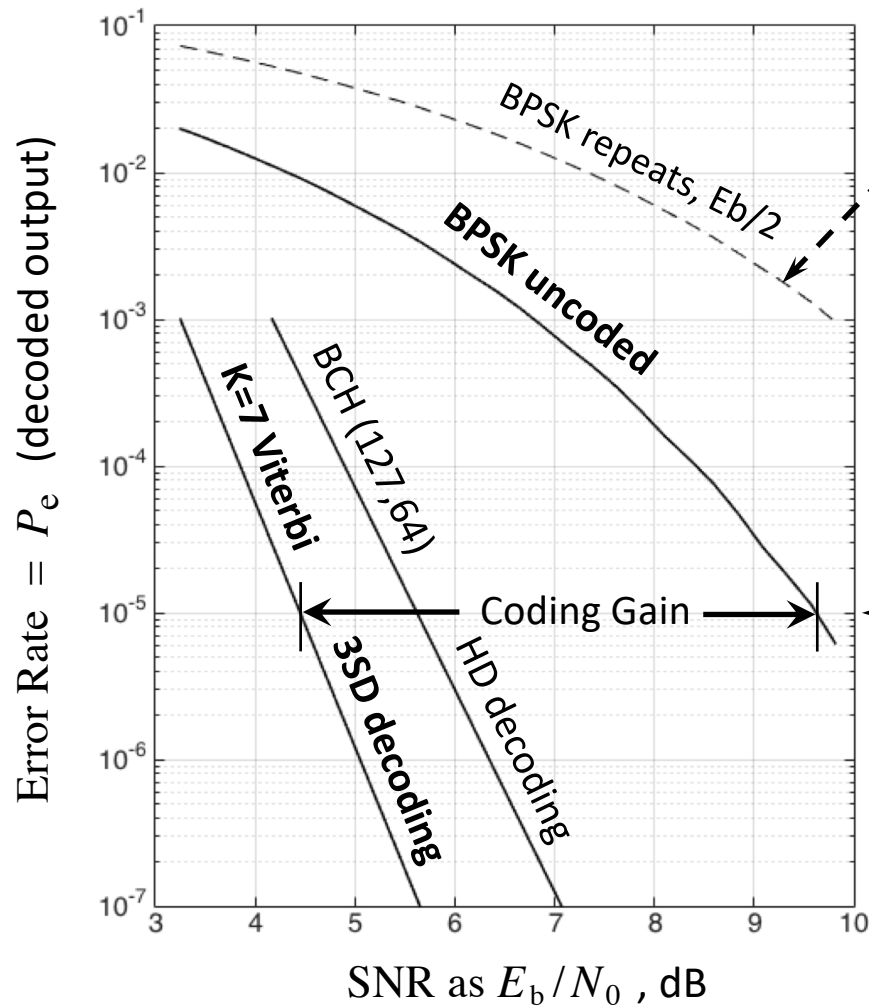
- a) Codes & the Viterbi Algorithm (VA) Decoder  
*and*
- b) Code Bounding Theory – Why It Works

SO, a Jump Point is provided to bypass the Theory.

After all, simulation and trials prove the VA works as advertised, and many are interested first and foremost in how it works as an algorithm and its implementation.

# Block vs Convolutional

## A Brief History of Redundancy at rate 1/2



Huge Post-Detection 2-Symbol Combining Loss

$\left(\frac{1}{2}\right) 2 p_2 (1 - p_2) + p_2^2 = p_2 \gg p_1$  : Coding Requires Distance!

Pre-Detection Combining recoups performance in special cases like fading & jamming

Diversity Schemes fall in between and can legitimately improve on uncoded  $P_e$

Majority & Hamming codes enticed theorists

Then appeared Golay's code, then BCH, then R-S

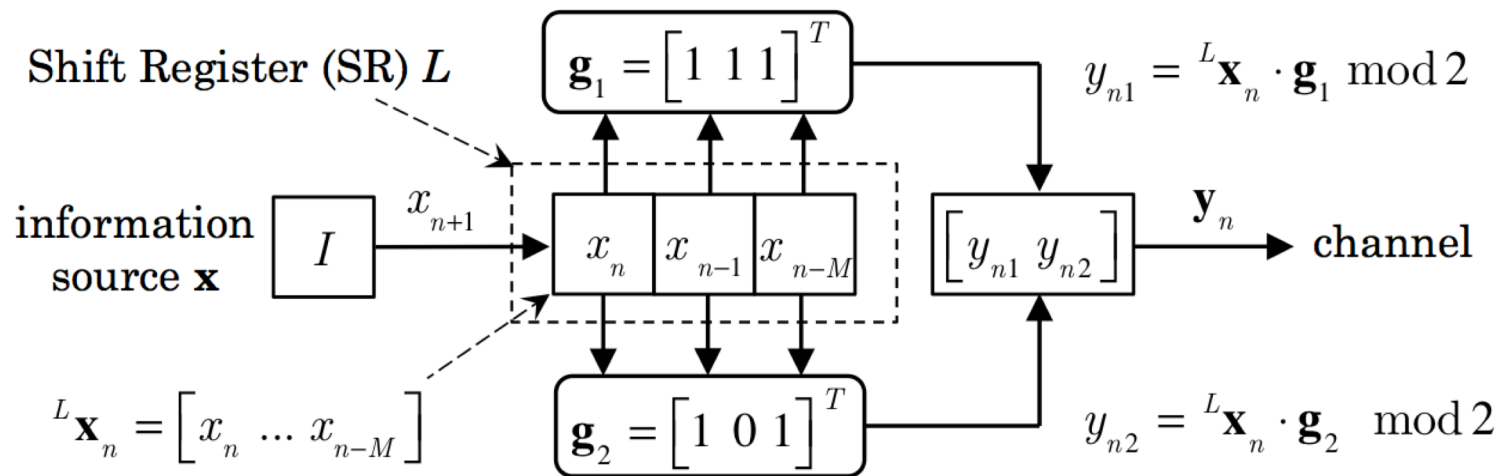
( $G_c$  indicated here for Convolutional / Viterbi)

Most of the Viterbi advantage shown here is in soft decisions, 3(-bit)SD, but not all of it!

A SATCOM rule-of-thumb goes: 1 dB gain (i.e., 25%) trades with 1/5<sup>th</sup> of Total System Cost

# Convolutional Codes

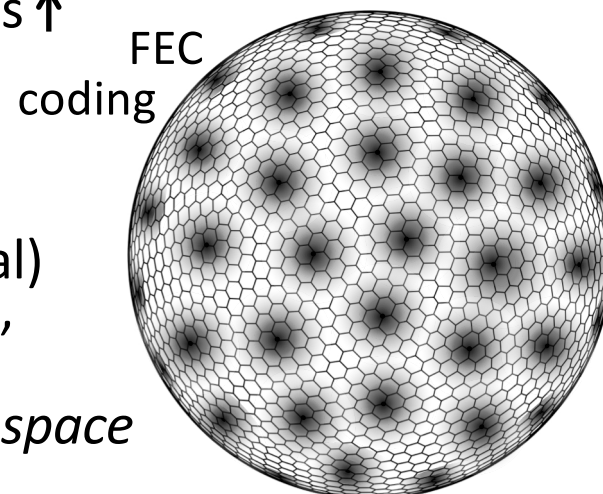
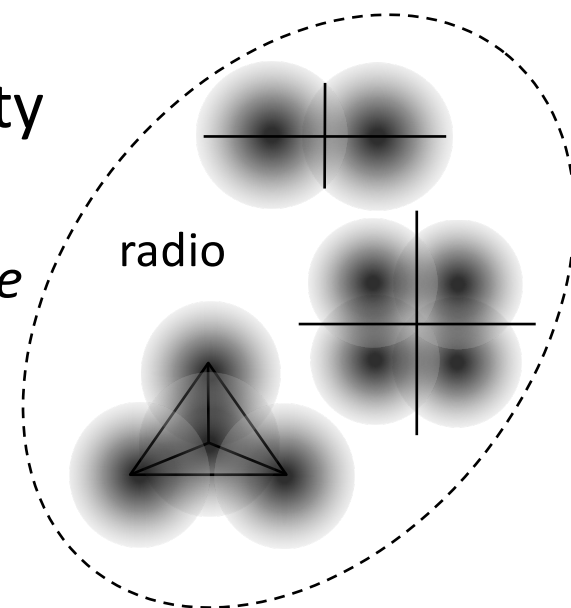
- So-named as code bit encoding superficially resembles FIR filtering
  - Hence “convolving” info with code; was a weak analogy, but it stuck
- Simple to build, fast with little complexity
  - Perfect for deep space missions, but also small terminals back in the day
  - Here’s the  $K = 3$ , code rate  $\frac{1}{2}$  canonical example



Codewords are “embedded” in code symbol sequence  $\mathbf{y}$  ( $= \mathbf{y}_1 \ \mathbf{y}_2 \ \dots \ \mathbf{y}_n \ \dots$ ) and occupy a *higher dimensional space* (than that of information sequence  $\mathbf{x}$ ) !

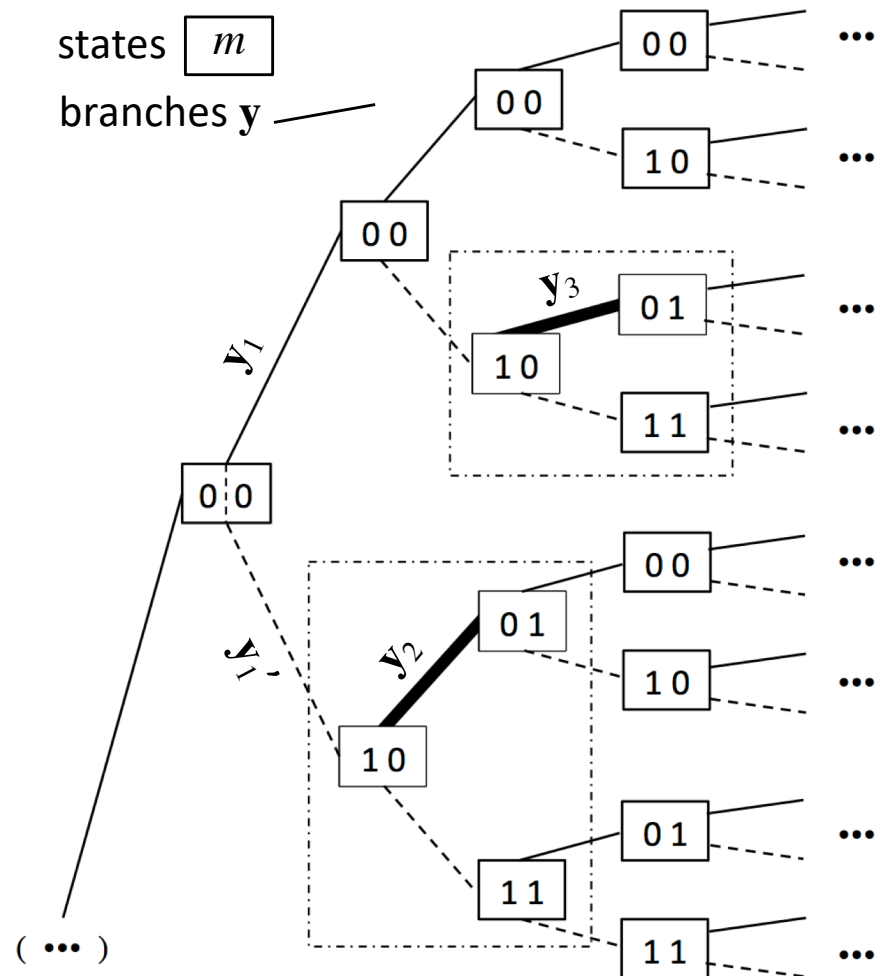
# Why is Dimensionality Such a Big Deal?

- Coding is all about increasing dimensionality
  - But hey, so was higher order modulation!
  - Pushing up  $R_0$  in pre-decision, radio *signal space*
- But FEC coding is *post*-decision!
  - It's about datasets in *information space*
  - Codewords lie on vertices of hypercubes
  - *Density* of used vertices goes  $\downarrow$  as dimensions  $\uparrow$
  - ***Distance between codewords*** climbs!
  - Could be “hard decision” (typ. of algebraic)
  - Could be “soft decision” (typ. of convolutional)
  - High dimensionality starts looking “spherical”
  - There's still an  $R_0$ , but now it's for *codeword space*



# 1966

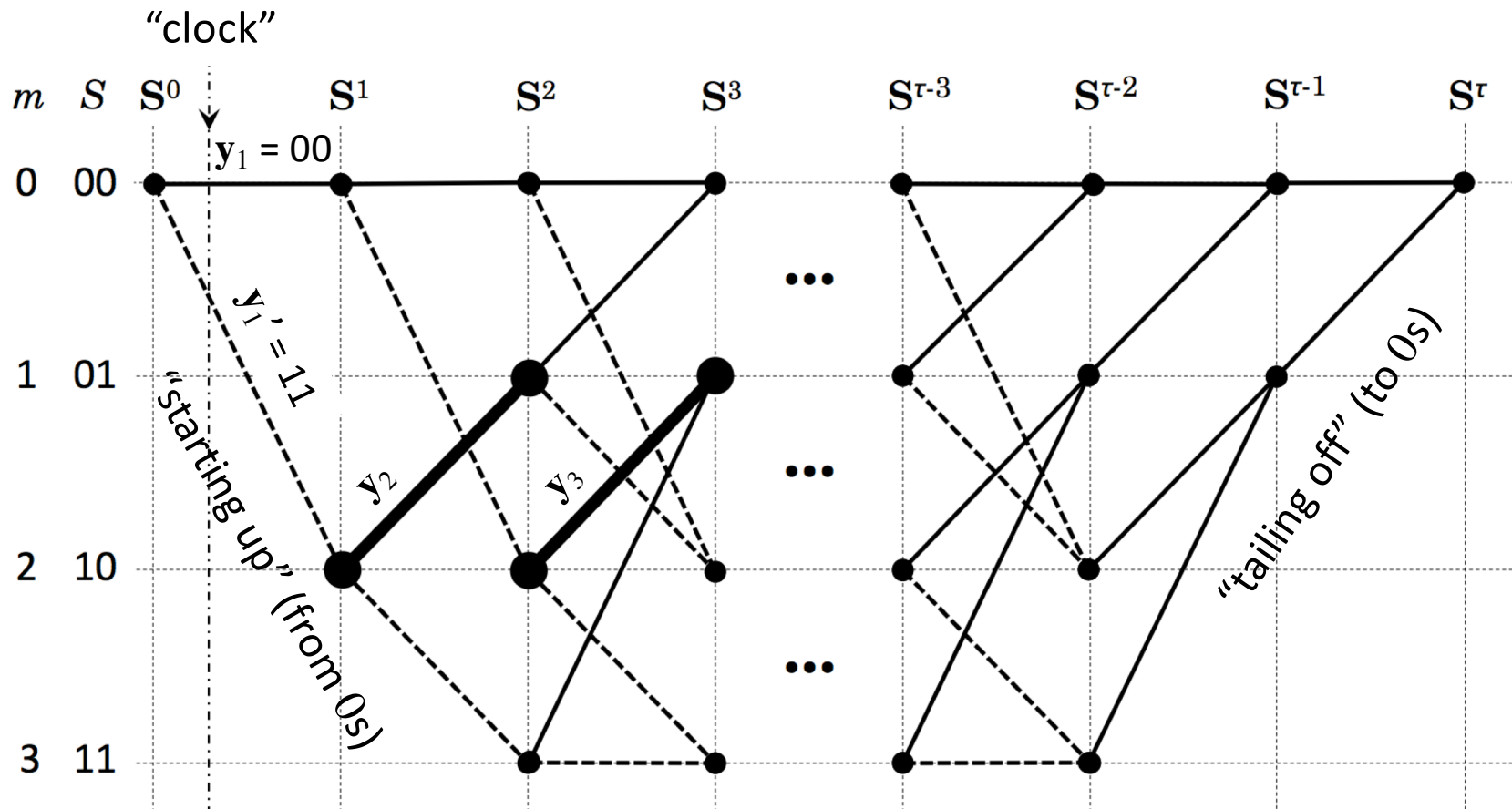
- Convolutional codes existed
  - Typical long *constraint length*  $K$
  - They could be almost capacity-approaching (but not in practice)
- Tree decoding ruled
  - Threshold search, Fano algorithm
  - LIFO buffer memory
  - Computing clock speed-up factor
  - $R_0$  = “computational cut-off” rate
- Decoding time is a r.v.
  - Decoding errors by buffer failure
  - Partial tree search was typical
  - Not maximum likelihood





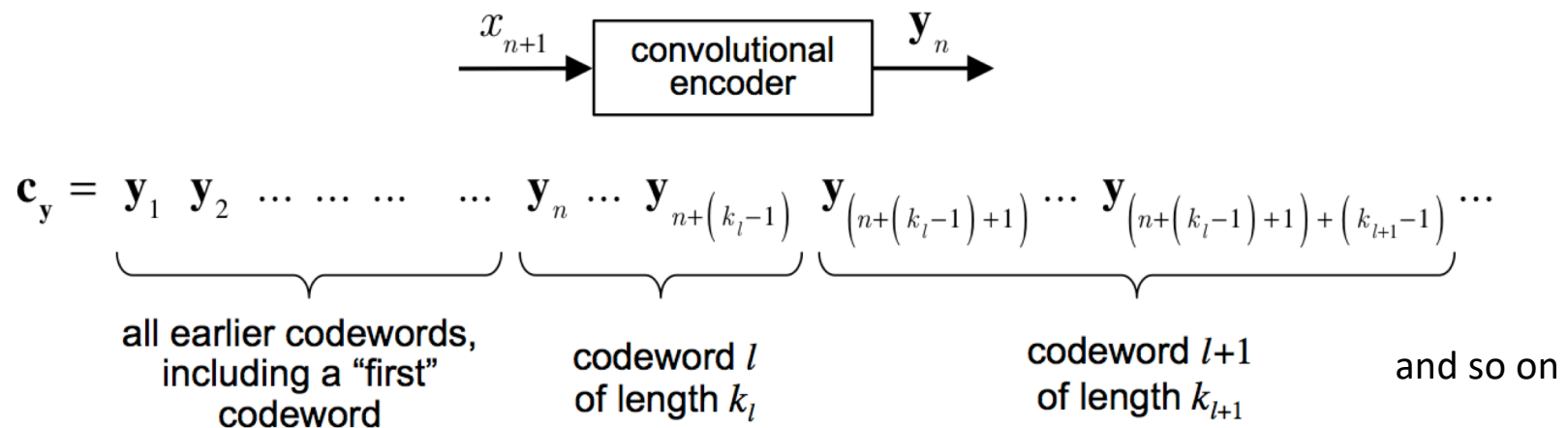
# The Trellis, A Bonsai'ed Tree

- Key is seeing code trees as sequences of encoder states  $S$ , or  $m$ 
  - Articulating Viterbi's move away from tree decoding
  - Same two example "branches" ( $y_2, y_3$ ) as for previous tree (**bold**)



# Convolutional Codewords

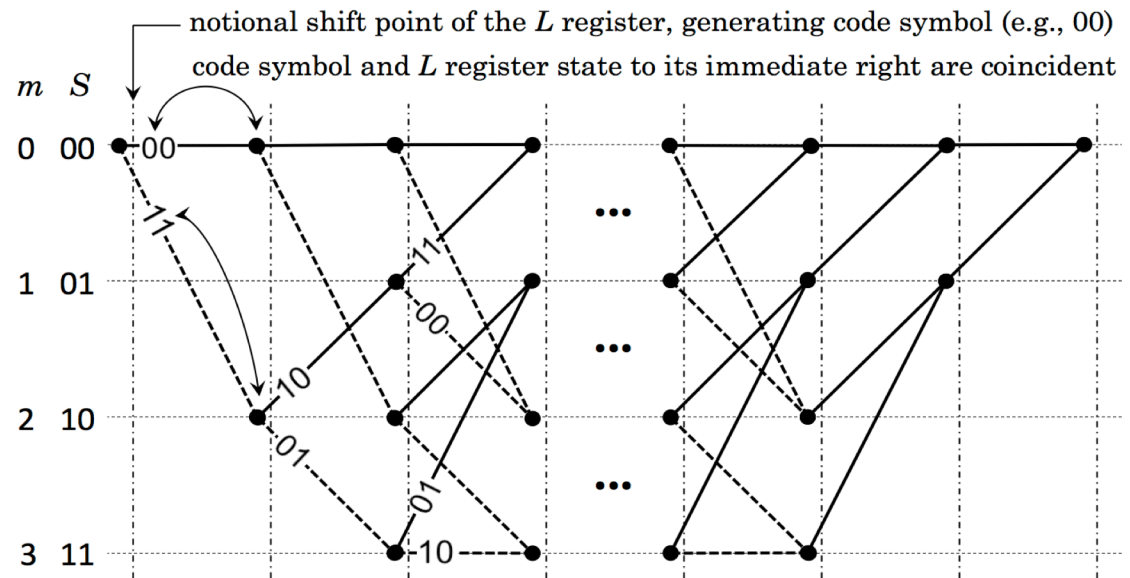
- Differing from (fixed length) Algebraic Code codewords
- Codewords not often discussed, but essential to theory
- “First” codeword composed of  $(K-1)/R_c$  code symbols  $\mathbf{y}$
- Each of a length defined by converging with (at some state  $m$ ) any other possible codeword of the same length
- Therefore, belonging to a *distribution* of codeword lengths



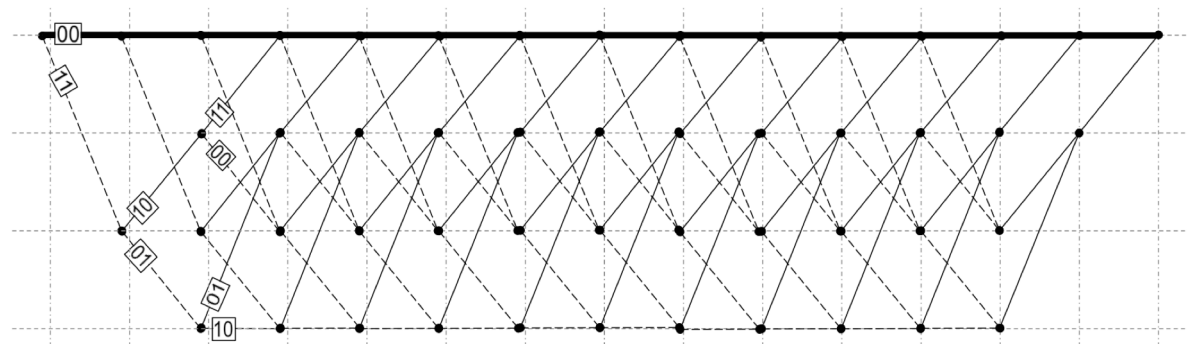
# Trellis Example, $K = 3, R_c = \frac{1}{2}, \{[111]^T, [101]^T\}$

Transmit Time = Trellis Depth →

By States and Branch  
Code Symbols  
but showing no paths  
just yet!

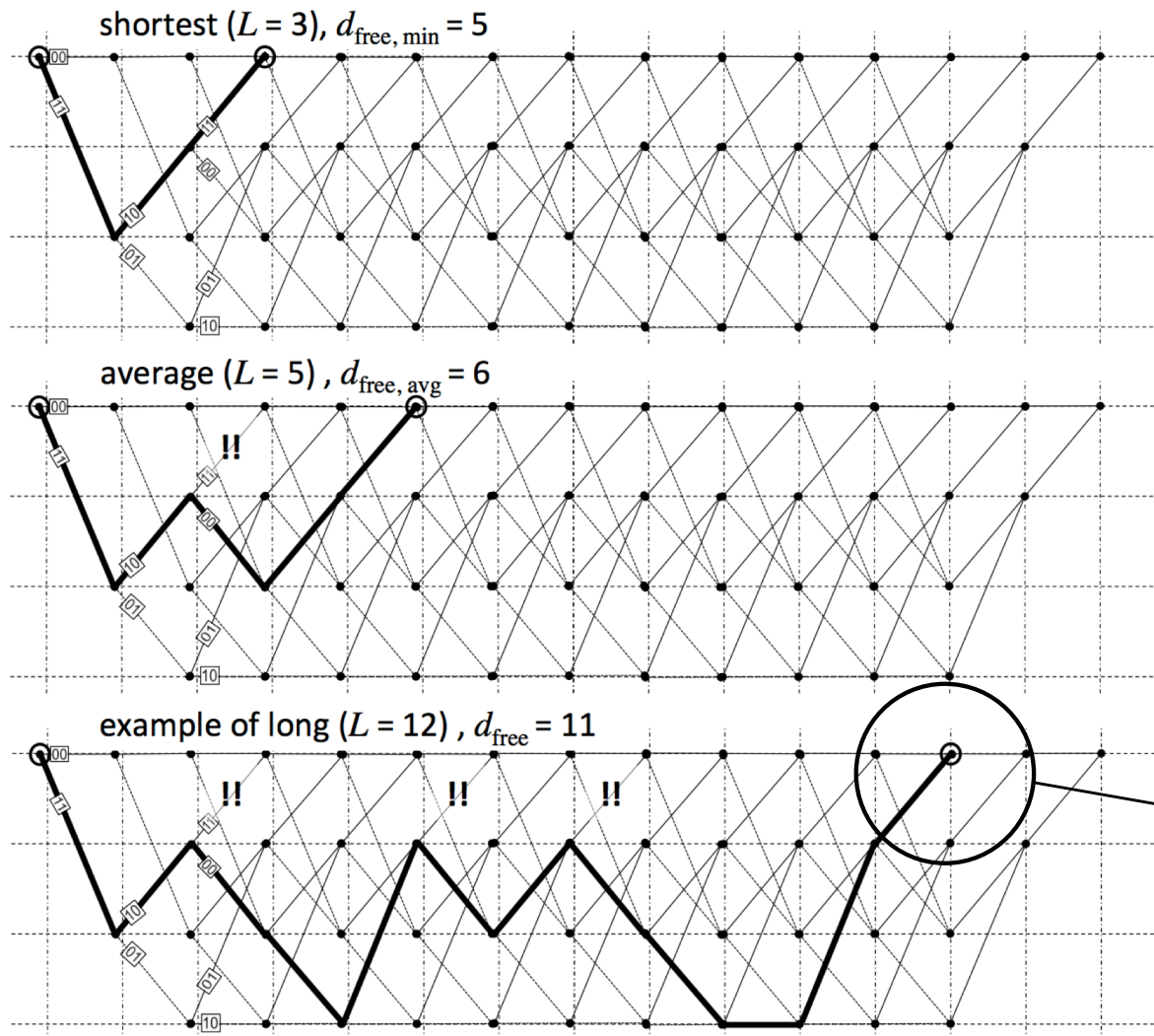


Canonical All-Zeros  
Codeword Path



# Some Codewords for $K=3, R_c = \frac{1}{2}$ Example

## Each Converging with, at least, an All-0s Codeword



By Code Linearity

Any of these Codewords need be compared *only* with an All-0s Codeword of equivalent length

Doing that, we get the “free” distance of the Codewords, and the minimum free distance

A blow-up of such an example region will display the behavior of VA state selection

# What Does the Game Plan Leave Out?

- Time is Limited: What are we choosing to leave out?
  - Encoder State Diagrams
  - Code Generator Functions
  - Catastrophic codes
  - Systematic vs Non-Systematic Codes
  - Non-binary & Multi- $K$  Convolutional Codes
  - Bandwidth Efficient Codes
  - Interleaving
  - Concatenated Codes
  - BCJR Theory
- However, a Simplest (Canonical) Case Demonstrates It All
  - Our friendly Rate  $\frac{1}{2}$ ,  $K = 3$  Binary Convolutional Code

# 1967

- A. J. Viterbi quietly publishes a paper (written in 1966)
  - On error bounds for convolutional codes
  - *and* an asymptotically optimum decoding algorithm
  - with review help from a computer scientist, L. Kleinrock
- Viterbi employed a very “tight” *lower* block bound on  $P_e$ 
  - due to Shannon, Gallager and Berlekamp
  - *Upper* bounds remained random coding type (Shannon capacity  $\mathcal{C}$ )
  - $R_0$  took on a new meaning from uncoded & sequentially-decoded
  - The new lower bound suggested useful decoding at  $R_0 \leq R < \mathcal{C}$
- Viterbi’s algorithm employed *state* searches
  - to *decide* among blocks of state sequences across trailing channel times
  - No mention of *trellises*, but strong hints (by precisely describing them)
- It would take another 2-3 years to “modernize” Viterbi Decoding
  - and another 12 years to publish “all of it” in a textbook

# Jump Point (for Bypassing Theory)

- You Really Only Need Know the Following
  - to be Aware of Viterbi Decoding's Power & Enduring Appeal
- 1. Key Difference between VA and Earlier Decoding
  - Viterbi's algorithm employed searches over *all states* to *decide* among adversarial candidate blocks of *state sequences*
- 2. Consequence of the Convolutional Coding Theorem
  - Viterbi-decoded convolutional codes beat the best "block" codes *given* "single pass" decoding, i.e. one-shot use of channel data
- 3. Conclusions of the Theory & "Genius" of the Algorithm
  - For that, go to

Jump Reentry is at Slide 28

# Viterbi's 1967 Paper

- First, everything is in *nats*
  - It could have been done in *bits*, BUT our fate was sealed
  - $e^{-\exp}$  forms are about the only  $D$ (istribution)s easily manipulated !
- $\ln M = \log_b M \ln b \dots$  so  $R_T = \ln M / T_M$  nats/s =  $\ln 2 / T_b$  bps
  - Keeping just that in mind, the paper becomes quite readable !
- Viterbi doesn't bother with "simple" limits on capacity  $\mathcal{C}$ 
  - like  $\mathcal{C} = H_{\max} = B \log_2(1 + C/N_0)$  and  $H_{\max, B \rightarrow \infty} = (C/N_0) / \ln 2$
  - Rather he asks, how closely can a code *approach*  $\mathcal{C}$  with  $B \sim k R_T$  ?
- It's about upper and lower *bounds* on convolutional codes
  - Specifically, a lower bound that closely approaches an upper bound
- **And an algorithm that meets the requirements of both**



# Bounds

- Coding bounds are on *decoded error probability*
  - Not on rate, code rate or bandwidth, etc. ... that comes later
- “Block” coding bounds came first and last!
  - Viterbi used block bounds “stretched” to a first *tight* lower bound on convolutional codes ... and it would produce a very different result !
  - To do this, Viterbi stood on the shoulders of R. G. Gallager
- Upper bounds (u.b.) came from Shannon’s random coding
  - A huge implication: *only the best “fixed” codes can approach u.b.*
- Lower bounds (l.b.) evolved from “sphere packing”
  - In block coding & the *very idea* of tightness (l.b. approaches u.b.)
  - Viterbi exploited Gallager’s function to upwardly *tighten* Shannon’s l.b.
  - It would involve *a lot* of math – read first half of textbook
- Bounds are all about proving that *some compliant* code exists!
  - Once in existence, it can be found and exploited

# Upper Bounds, 1

- Starts out simple and with tweaks becomes “tighter”

simple union bound

$$P_{E_m} = \Pr\{\mathbf{y} \in \bar{\Lambda}_m \mid \mathbf{x}_m\} = \Pr\left\{\mathbf{y} \in \left(\bigcup_{m' \neq m} \Lambda_{mm'} \mid \mathbf{x}_m\right)\right\} \leq \sum_{m' \neq m} \Pr\left\{\ln \frac{p_N(\mathbf{y} \mid \mathbf{x}_{m'})}{p_N(\mathbf{y} \mid \mathbf{x}_m)} \geq 0 \mid \mathbf{x}_m\right\} = \sum_{m' \neq m} P_E(m \rightarrow m')$$

refining the kernel with a *bifurcation* trick

$$P_E(m \rightarrow m') = \sum_{\mathbf{y}} f(\mathbf{y}) p_N(\mathbf{y} \mid \mathbf{x}_m) \quad f(\mathbf{y}) = \begin{cases} 1 \leq \sqrt{p_N(\mathbf{y} \mid \mathbf{x}_{m'}) / p_N(\mathbf{y} \mid \mathbf{x}_m)} & \mathbf{y} \in \Lambda_{mm'} \\ 0 \leq \text{same!} & \mathbf{y} \notin \Lambda_{mm'} \end{cases}$$

gets you Bhattacharyya’s (union) bound for one (of multiple) transitions

$$P_E(m \rightarrow m') \leq \sum_{\mathbf{y}} \left\{ p_N(\mathbf{y} \mid \mathbf{x}_{m'}) p_N(\mathbf{y} \mid \mathbf{x}_m) \right\}^{1/2} \quad \mathbf{y} : \mathbf{y} \in (\Lambda_{mm'} \cup \bar{\Lambda}_{mm'}), \forall \mathbf{x}$$

## Upper Bounds, 2

- Bhattacharyya's unique transition bound gets tweaked by Jensen's Inequality via "tightening parameters"  $\lambda$  and  $\rho$
- Reducing a ratio which is already *intrinsically*  $< 1$

Jensen's Inequality  
plus convexity,  
the two biggest  
tools in the box

$$f(\mathbf{y}) \leq \left\{ \sum_{m' \neq m} \left[ \frac{p_N(\mathbf{y} | \mathbf{x}_{m'})}{p_N(\mathbf{y} | \mathbf{x}_m)} \right]^\lambda \right\}^\rho \quad \text{all } \mathbf{y} \in Y_N, \rho > 0, \lambda > 0$$

and again by bifurcation

allowing Gallager's bound, now over *any*  $m$  and with a near magical choice of  $\lambda$

$$P_{E_m} \leq \sum_{\mathbf{y}} \left[ p_N(y | \mathbf{x}_m) \right]^{1/(1+\rho)} \left\{ \sum_{m' \neq m} \left[ p_N(y | \mathbf{x}_{m'}) \right]^{1/(1+\rho)} \right\}^\rho \quad \rho > 0 \quad \text{and } \lambda = 1 / (1 + \rho)$$

## Upper Bounds, 3

- Average of Gallager's (union) bound over *all* messages
- Becoming quite powerful for discrete *or* continuous channels
- Ends up bounding  $R < R_0 = E_0(1, \mathbf{q})$  but tighter\* for  $\rho < 1$


remember, *nats* !

$$\bar{P}_{E_m} < M^\rho \sum_{\mathbf{y}} \left[ \sum_{\mathbf{x}} q(\mathbf{x}) p_N(\mathbf{y} | \mathbf{x})^{1/(1+\rho)} \right]^{1+\rho} = e^{-N[E_0(\rho, \mathbf{q}) - \rho R]} \quad 0 \leq \rho \leq 1$$

using Gallager's function  $E_0(\rho, \mathbf{q}) \equiv -\ln \sum_{\mathbf{y}} \left[ \sum_{\mathbf{x}} q(\mathbf{x}) p_N(\mathbf{y} | \mathbf{x})^{1/(1+\rho)} \right]^{1+\rho}$

\* obsoleting our old W&J friend  $\bar{P}_{E_m} < 2^{-N[R_0 - R_N]}$  (in bits, of course)


# Lower Bounds, 1

- *Fundamentally,*
  - ALL the bounds come from Neyman-Pearson & Chernoff!
    - Can be side-stepped for upper bounds (u.b.) by way of Unions
    - But the best minds resort to N-P&C for lower bounds (l.b.)
    - l.b.s nicely captured by Fano's argument called "sphere-packing"
    - It all comes from convexity of the log of all probability combinations
- Viterbi nails l.b.s in Chapter 5 of his 1979 text
  - Comes down to diverging paths lower-bounding decoding errors
    - These "divergers" become the famous "mergers" of Trace Back 
  - Convolutional code "stretching" and "overhead" of blocks is key
- l.b.s take a back seat to u.b.s in all that follows
  - For this reason, and the great complexity of lower bound theory ...
- We'll just sketch the route to l.b.s here

## Lower Bounds, 2

- Neyman-Pearson, two hypotheses  $a$  and  $b$

$$P_a \equiv \sum_{\mathbf{y} \in Y_b} p_N^{(a)}(\mathbf{y}), \quad P_b \equiv \sum_{\mathbf{y} \in Y_a} p_N^{(b)}(\mathbf{y}), \quad Y_a = \bar{Y}_b \text{ and vice versa}$$




$$\mu(s) \equiv \ln \sum_{\mathbf{y}} p_N^{(a)}(\mathbf{y})^{1-s} p_N^{(b)}(\mathbf{y})^s, \quad 0 \leq s \leq 1 \text{ and convex}$$

$$P_a \leq e^{\mu(s) - s\mu'(s)}, \quad P_b \leq e^{\mu(s) - (1-s)\mu'(s)} \quad \text{by Chernoff bounds}$$

- Now, make two hypotheses, i.e. two codewords! (Gallager)

$$\mu(s) \equiv \sum_{n=1}^N \ln \left[ \sum_{y_n} p(y_n | x_{an})^{1-s} p(y_n | x_{bn})^s \right], \quad 0 \leq s \leq 1 \text{ for two codewords } x_{an} \text{ and } x_{bn}$$



l.b.

$$e^{\mu(s_0) - N\phi(s_0)} < P_x < e^{\mu(s_0)}, \quad x = a \text{ or } b \text{ iff } P_a = P_b \text{ s.t. } \mu'(s = s_0) = 0$$

## Lower Bounds, 3

- Exploiting a “*tour de force*” sphere-packing (*sp*) approach
  - Viterbi shows, unsurprisingly exponential in form, but in base 2 (!)

bit error rate is lower bounded by  $P_b \geq 2^{-Kb(E_{csp}(R) + o(K))/R}$

where

$$E_{csp}(R) = E_0(\rho), \quad 0 < \rho < \infty ; \quad R = E_0(\rho) / \rho, \quad 0 < R < C$$

- In effect, this “squeezes” the l.b. toward an u.b.
  - Viterbi shows the two actually converge for  $E_0(1) = R_0 < R < C$
  - Meaning, in short, that the Convolutional Coding Exponent is as good as we can do (at least for single-pass FEC)
  - But, a strong suggestion here that  $R \rightarrow C$  with the right scheme!

# Summarizing Viterbi's 1967 Paper

- He uses  $E_0(\rho, q)$  to map  $R_T$  across the domain  $0 < R_0 < \mathcal{C}$ 
  - it would be finally expressed (in his 1979 text) as the *Convolutional Channel Coding Theorem*
- He finds two algorithms that adhere to the bounds
  - the *nonsequential* one leads to the modern Viterbi Algorithm
  - a “semisequential” variant is computationally too heavy
- The nonsequential algorithm
  - bears resemblance to simplex algorithm search for maxima
    - but unlike simplex, it may increase or decrease along the way
  - exploits dynamic programming ideas (Richard Bellman?)
    - cyclic iteration, forwarding of previous calculations
    - pre-computing branch metrics (even if only implied)
    - these now-simple ideas were revolutionary at the time



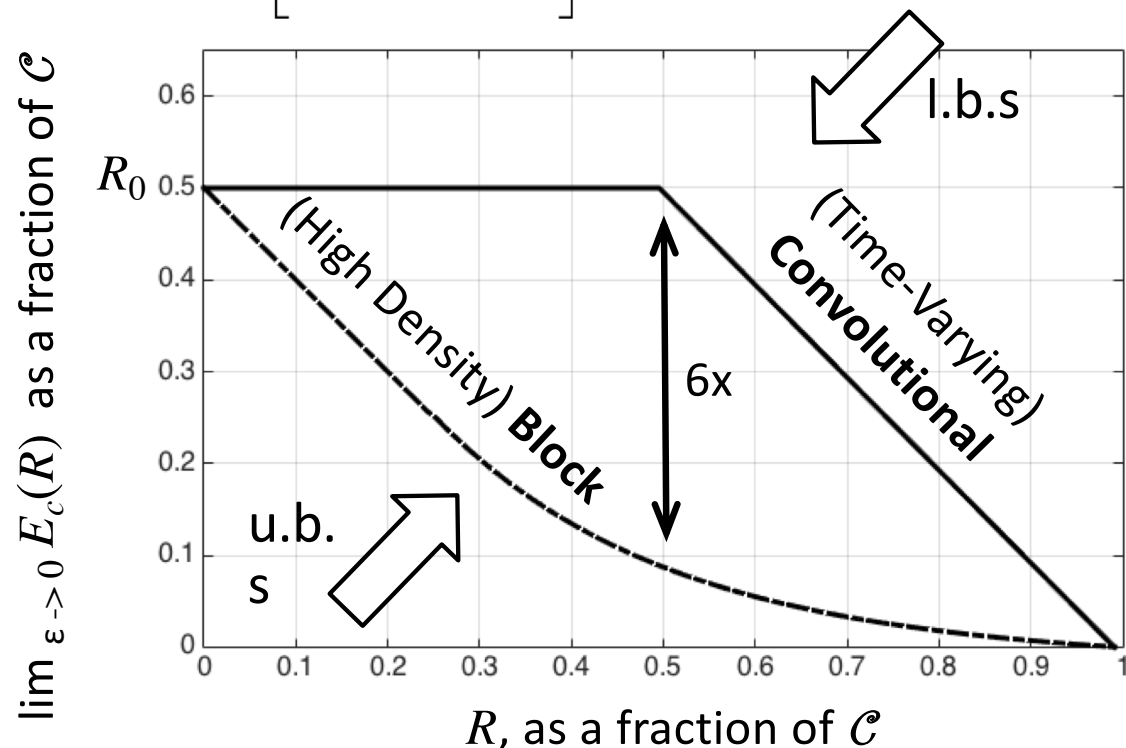
# Convolutional Channel Coding Theorem, 1

For any DMC, there is a time-varying convolutional code of constraint length  $K$ , rate  $b/n$ , and an arbitrary block length whose  $P_b$  in (single pass) ML decoding (over all possible codes) is upper bounded by

$$P_b \leq \bar{P}_b \leq \frac{LE[n_b(j)]}{Lb} < (2^b - 1) \frac{2^{-KbE_c(R)/R}}{\left[1 - 2^{-\varepsilon bE_c(R)/R}\right]^2}, \quad 0 < \varepsilon < 1$$

For a *very noisy channel* (sort of a worst, but also a practical case) here was the tantalizing result

Also, the first hint of turbo codes (!)



# Convolutional Channel Coding Theorem, 2

What's this Time-Varying (Code) thing, and does it matter?

All practical convolutional codes (I know of) are Time-Invariant (meaning their multiplier polynomials never change)!

Time-Varying was simply required by Viterbi's mathematical bounding strategy, based on random codes – including “poor” codes. But it doesn't matter.

A “good” code is so much better than average, and information is so random (if only scrambled), that a best, or even good, Invariant Code is close to optimum.

What's this High Density thing, and does it matter?

Practical “block” codes (up to the mid-1990s) had “high density” parity check matrices. Gallager had invented a “low density” PCM (LDPC\*) code in 1961, but no one knew how to decode it in a practical way (that would wait until 1994)!

So the only real competitor to convolutional in 1967 was “block” coding, e.g. BCH and R-S codes, and some bespoke ones like “burst error correcting” codes.

\* Gallager *did* actually call his capacity-approaching codes LDPC in a 1962 paper

# Convolutional Channel Coding Theorem, 3

But there was a problem: the u.b. is too loose at low rates! To wit

$$P_b < 2^{-(K-2)}, \quad \varepsilon = 1, \quad b = 1, \quad E_c(R) = R = R_0 = C/2$$

One way to improve is to get rid of “bad” codes, a.k.a. “expurgation”

Gallager (the usual suspect) leads with his Expurgated Coding Theorem

$$P_{E_m} < e^{-N E_{ex}(R)}, \quad 0 \leq \rho \leq 1 \quad m, \text{ a code from } m \in M(N)$$

where

$$E_{ex}(R) = \max_{\mathbf{q}} \sup_{\rho \geq 1} \left[ E_x(\rho, \mathbf{q}) - \rho \left( R + \frac{\ln 4}{N} \right) \right]$$

where the “expurgated version” of Gallager’s function is

$$E_x(\rho, \mathbf{q}) = -\rho \ln \left\{ \sum_x \sum_{x'} q(x) q(x') \left[ \sum_y \sqrt{p(y|x) p(y|x')} \right]^{1/\rho} \right\}, \quad \rho \geq 0$$

# Convolutional Channel Coding Theorem, 4

Expurgating, a powerful case (Viterbi & Odenwalder, 1969) for  $R \leq R_0$  exploits

$$E_{cex}(\rho, \mathbf{q}) = \max_{\mathbf{q}} E_x(\rho, \mathbf{q}), \quad 1 \leq \rho < \infty$$

For any BSC, there is a time-varying convolutional code of constraint length  $K$ , rate  $b/n$ , and an arbitrary block length whose  $P_b$  in (single pass) ML decoding is bounded by

$$P_b < \left[ \frac{2^b - 1}{(1 - 2^{-b\varepsilon})^2} \right]^{E_{cex}(R)/[R(1+\varepsilon)]} 2^{-KbE_{cex}(R)/R}, \quad 0 < \varepsilon$$

Things are getting much better! To wit, a  $K = 7$  exponent jumps from 5 to 9!

$$P_b < 2^{-(3/2)(K-1)}, \quad \varepsilon = 1, \quad b = 1, \quad E_{cex}(R) = (3/2)R_0, \quad R = R_0 = C/2$$

# Convolutional Code Theory, Put to Bed

- Error Rate Bounds Were Important for Several Reasons
  1. Put Conv Coding on Solid Foundation – Confidence
  2. Encouraged Finding Codes & Algorithms to Meet Them
- Post 1967
  1. Encouraged Shifting from Block to Convolutional
  2. Encouraged Shifting from Sequential to Viterbi Decoding
  3. It was clear that an “outer” very-high-rate code could “clean up” an “inner” Viterbi decoder for *super-low* decoded error rates ... so-called concatenated coding

# The Genius of Viterbi's Algorithm, 1

- It achieves maximum likelihood (ML) decisions
  - with *minimal* computational complexity
  - ML compares against *every* possible information sequence
- Consider a code *tree* of  $N$  code symbol branches
  - or  $N$  information bits if  $q = 2$  (binary information input to encoder)
  - total number of possible paths is  $2^N$
  - with some “reuse,” total possible (exhaustive) “calculations” is  $2^{N/2}$
- Consider same code as a *trellis*
  - total number of possible *adversaries* to the set of states is  $2^{K-1}$
  - total exhaustive “calculations” is  $N 2^{K-1}$
- ML with “*only*”  $N = 32$  bits and  $K = 7$  coding
  - $\sim 6 \times 10^4$  calcs as a tree, but just  $2 \times 10^3$  calcs as a trellis
  - ratio grows as  $2N 2^{N/2-K}$  ... quickly going out of sight with  $N$

# The Genius of Viterbi's Algorithm, 2

- Like Sequential Decoding, Viterbi handles Soft Decisions
  - exploiting them deftly, with minimal computational complexity
  - also handling erasure and other memoryless channel models
- It outputs Concentrated Bursts of Decoded Errors
  - most unlike the long codeword errors of algebraic codes
  - perfect for concatenated coding “clean up”
- It has a clean “growth path” to more powerful codes
  - only quadruples in complexity for a full  $\frac{1}{2}$  dB of added coding gain
  - yes, you begin to inch up toward Capacity (but turbo would leap)
- It has a very simple encoder
  - perfect for spacecraft, beacons and throwaway devices
  - even a sequential decoder requires a lot more encoding complexity

# The Genius of Viterbi's Algorithm, 3

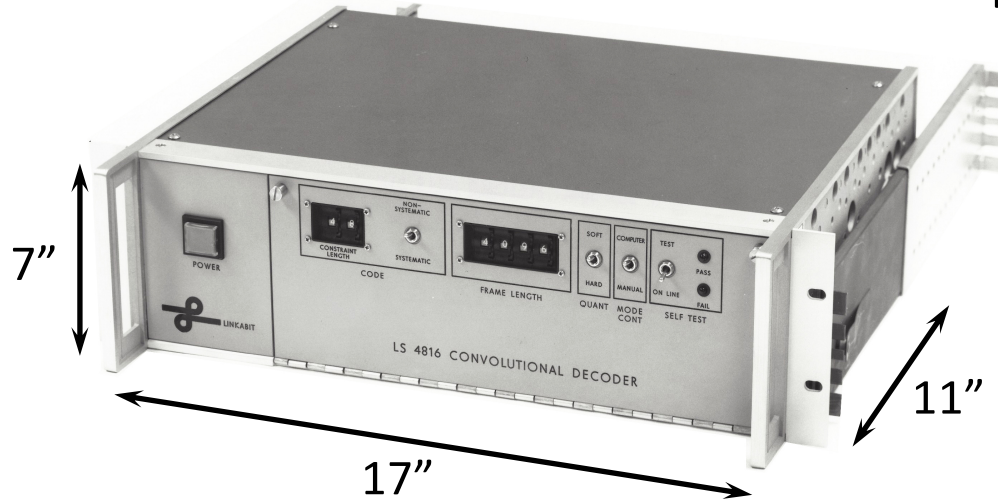
- It was natively Continuous Data Capable
  - it was even “self-starting” (not needing to find a block SOM)
- It was easily “Blockened” ... Viterbi anticipated Packets
  - initially, just zero'ing (both ends)
  - but later including efficiency refinements like “tail biting”
- It led easily to Soft Output Viterbi Algorithm (SOVA)
  - setting up iterative decoding before it was even discovered
- It could support *trellis and multi-h (de)modulation*
  - the first big jump for phoneline modems, circa 1984
- It was versatile
  - found other applications
  - in speech recognition, imperfect channels (ISI etc.)



# 1970s

## Linkabit\*, almost the only game in town

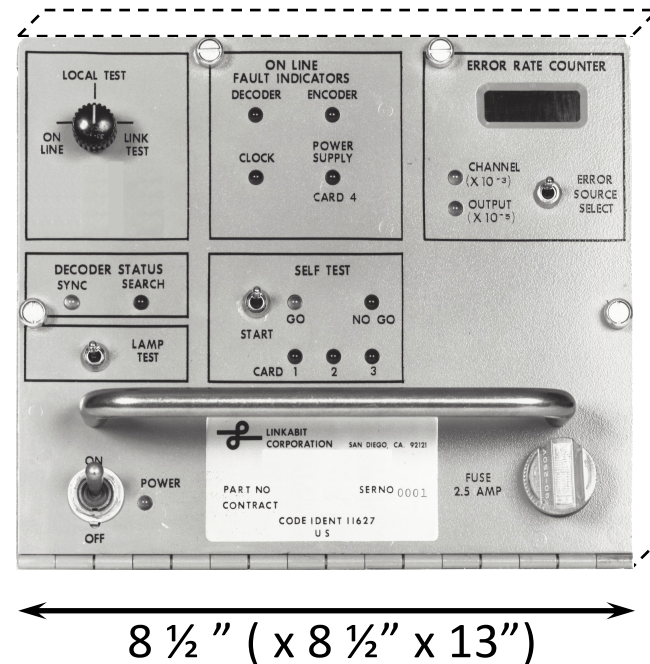
- Two examples of production convolutional decoders
  - S/LS TTL logic and Bipolar RAM, Soft Decisions of course



LS4816 Sequential Decoder - **tree**

$R = 1$  Mbps,  $K$  up to 48,  $R_c = \frac{1}{2}$  sys or non-  
~ 30 lbs, 50 W, \$5,000 (1977USD)  
became an LS56 LSI w/in 4 years

LV7017 Encoder/Viterbi Decoder - **trellis**  
 $R = 10$  Mbps,  $K = 7$ ,  $R_c = \frac{1}{2}$  non-sys  
~ 35 lbs, 85 W, \$15,000 (1977USD)

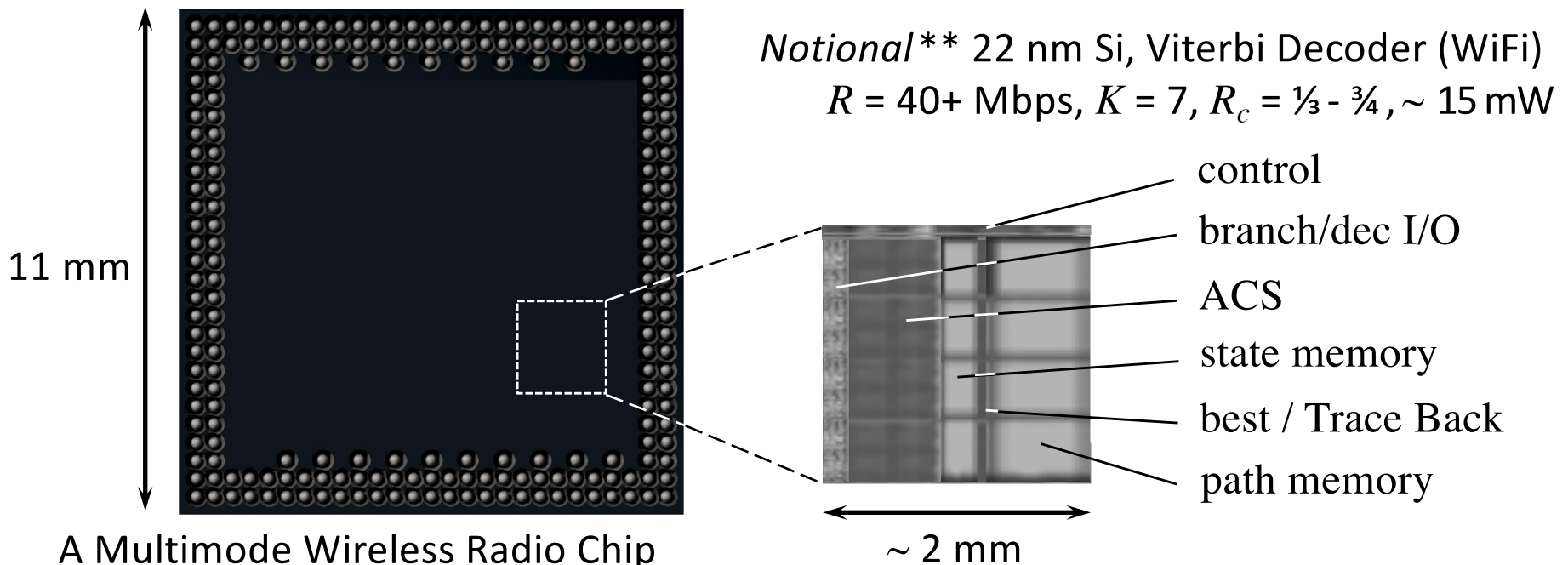


\* *That* Linkabit is long gone!

# 2010s

## Irwin Said It Best (about Moore's Law)

- “Don't worry about complexity ... the chips will come to us!”
  - consider an IoT SoC ASIC\* today ... 4x perf &  $10^5$  smaller vs LV7017



*Notional\*\** 22 nm Si, Viterbi Decoder (WiFi)  
 $R = 40+$  Mbps,  $K = 7$ ,  $R_c = \frac{1}{3} - \frac{3}{4}$ ,  $\sim 15$  mW

A Multimode Wireless Radio Chip

say, 4+ different waveforms

$\sim$  neg wgt, 150 mW, \$8 (est 1977USD)

**< \$0.50 (1977USD) decoder !**

\* Resemblance to any mfr, living or dead, is coincidental!

\*\* best guess

# 1979

- Viterbi (& Omura)'s text\* appears, again in *nats* (vice bits)
- 1967 paper is now but one of a 100 bibliography items
  - But it's results are better explained
  - Trace Back is just called Path Memory Truncation w/o elaboration
  - I.e., Intellectual Property knowhow was still in the driver's seat !
- Path Memory Truncation
  - Effectively, Fig. 5.6 would show about everything anyone needed to know about Trace Back
  - Except the mechanics of how to do it efficiently
  - Captured in specific methods like Chain Back discussed later on

\* The should-be-famous Principles of Digital Communication and Coding!

## The Algorithm, Part 1 ...

*There is a Correctly Encoded Trellis Path*

It is One Unique Path through the  
“Open” State Transition Diagram that is the Trellis

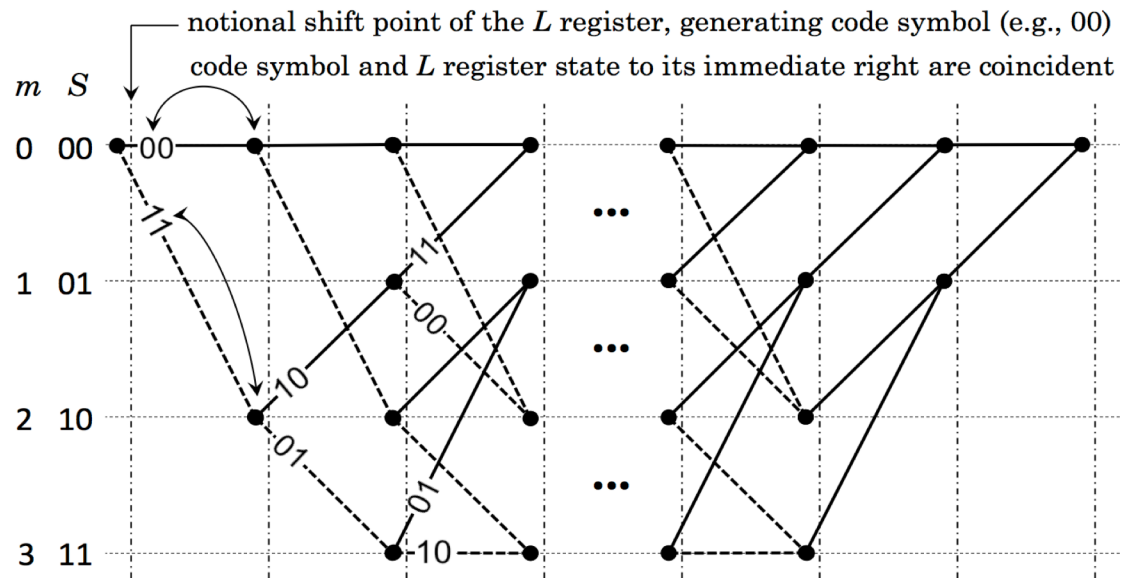
Its Discovery is the Goal  
of *Any* Decoding Algorithm,  
including the VA

# Trellis Example, $K = 3$ , $R_c = \frac{1}{2}$ , $\{[111]^T, [101]^T\}$

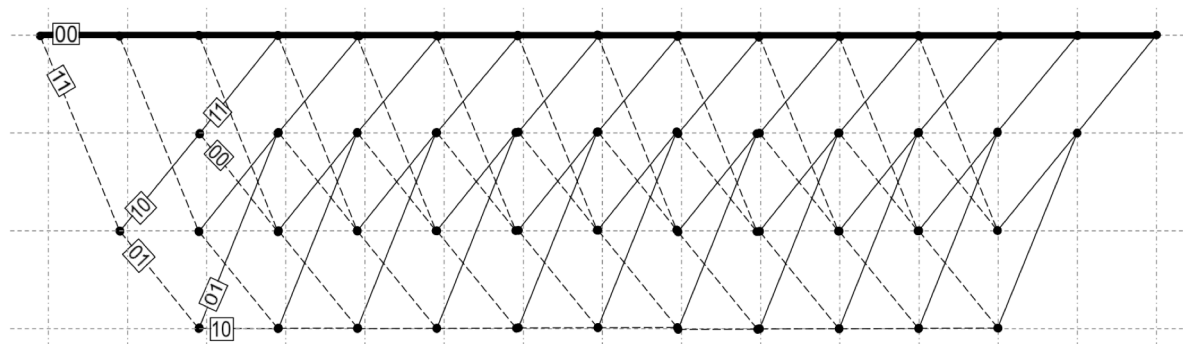
Trellis Time or Depth  $\longrightarrow$

By States and Branch  
Code Symbols

but showing no paths  
just yet!



Canonical All-Zeros  
Codeword Path

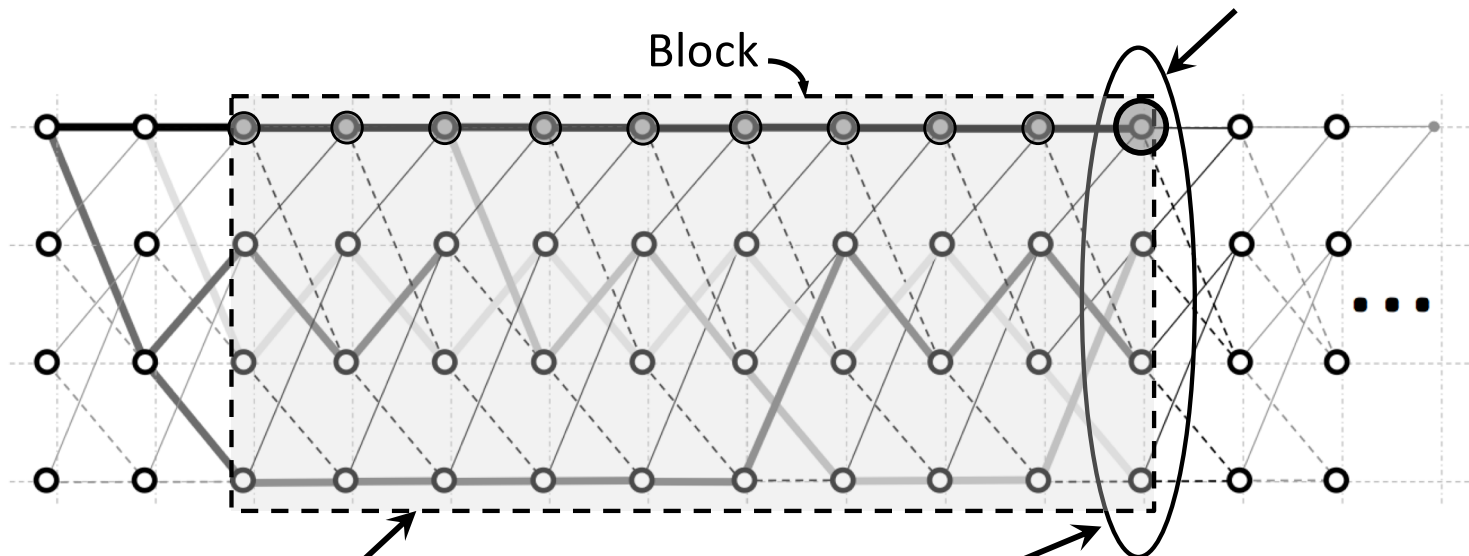


# 1967 Viterbi Paper Algorithm from 50,000'

A recent block of code symbols has been received

They have been mapped (by the algorithm) into possible *unique* paths

Each such path ends at one of the possible encoder states at  $t_{\text{now}}$



**To Decode:**

**First**, pick the *most likely* (i.e. “best”) state, e.g. ●, at  $t_{\text{now}}$

**Second**, from that best state at  $t_{\text{now}}$  *alone*, discern (i.e., decode) the underlying information *back* across the Block belonging to unique path (successively chosen as ●s by the algorithm) leading to that best state

The Algorithm, Part 2 ...

*There is a Best State*

Finding It is the Best Known Aspect of the VA

# Viterbi Algorithm State & Best State Selection

Overall, at Each Successive Trellis Depth, DO THIS

(what sets the Viterbi Algorithm apart from tree searching is its consideration of every state for every info bit ... the distinction of *maximum* likelihood decoding)

(0) For each of  $M$  possible encoder states of set  $M$ ,  $m \in M = \{m_0, m_1, \dots, m_{2^{K-2}}, m_{2^{K-1}}\}$ , each of  $K = 1 + \log_2 M$  length in bits, perform the following one full iteration.

For a hypothesized code symbol  $\hat{X}_t$  sent into the channel at  $t$ , a transition from a preceding state  $\hat{m}$  at time  $t-1$  to a current state  $m$  at  $t$ , select a best case for  $\{S_t = m\} \Leftrightarrow \{S_{t-1} = \hat{m}; \hat{X}_t(\hat{m}, m)\}$  for a one of the  $\hat{m} \in \{m', m''\}$  candidates.



# Viterbi Algorithm State & Best State Selection

## Steps 1, 2 – Evaluate Received Code Symbol

Obtain channel transition probabilities for each candidate  $\hat{m}$  “branching” to  $m$  of

(1)  $\Pr\{Y_t | \hat{X}_t(\hat{m}, m)\}$  and assign *branch metrics* for the two candidates,

(2)  $\lambda_t(m', m) \equiv \frac{\Pr\{Y_t | \hat{X}_t(m', m)\}}{\Pr\{Y_t | \hat{X}_t(m'', m)\}}$  and  $\lambda_t(m'', m) \equiv \frac{\Pr\{Y_t | \hat{X}_t(m'', m)\}}{\Pr\{Y_t | \hat{X}_t(m', m)\}}$ .

alternative branch “likelihood metrics,” or LMs

LMs are more convenient expressed as logarithms, or LLMs, since these can be added, avoiding multiplication

$${}^l_N \lambda_t(m', m) = \log_N \left( \frac{\Pr\{Y_t | \hat{X}_t(m', m)\}}{\Pr\{Y_t | \hat{X}_t(m'', m)\}} \right)$$

# Viterbi Algorithm State & Best State Selection

## Steps 3, 4 – The “ACS” Operation

Obtain the preceding *state metrics*  $\Lambda_{t-1}$  associated with each candidate branch,

(3)  $\Lambda_{t-1}(m')$  and  $\Lambda_{t-1}(m'')$ . Then for the prospective state metric  $\Lambda_t(m)$

(4) (a) Form the inequality metrics  $\Lambda_{t-1}(m') \cdot \lambda_t(m', m)$  ,  $\Lambda_{t-1}(m'') \cdot \lambda_t(m'', m)$ .

(4) (b) Compare, e.g. as  $\Lambda_{t-1}(m') \cdot \lambda_t(m', m) > \Lambda_{t-1}(m'') \cdot \lambda_t(m'', m)$  then

(4) (c) if inequality TRUE then select  $\hat{m} = m'$  ; else, select  $\hat{m} = m''$  .

alternative log LM math, without which ACS wouldn't be ACS !

if  ${}^{l_N} \Lambda_{t-1}(m') + {}^{l_N} \lambda_t(m', m) > {}^{l_N} \Lambda_{t-1}(m'') + {}^{l_N} \lambda_t(m'', m)$  , select  $\hat{m} = m'$  , else  $\hat{m} = m''$  ,  
 and then  ${}^{l_N} \Lambda_t(\hat{m}) = {}^{l_N} \Lambda_{t-1}(\hat{m}) + {}^{l_N} \lambda_t(\hat{m}, m)$  .

# Viterbi Algorithm State & Best State Selection

## Step 5 – Save (Path) Metric Result & Clean Up

for every state, a likelihood metric (LM) or, equivalently, its LogLM (LLM)

(5) (a) Assign  $\Lambda_t(m) \equiv \Lambda_{t-1}(\backslash m) \cdot \lambda_t(\backslash m, m)$  for  $\backslash m$  as selected and save.

(5) (b) Discard  $\Lambda_{t-1}(\backslash m)$  assigned in the previous iteration.

(5) (c) Discard  $\lambda_t(\backslash m, m)$ .

so seemingly boring a detail, except that it became a crucial refinement of the VA with trace back in place

# Viterbi Algorithm State & Best State Selection

## Step 6 – Save Best State (if so)

(6) (a) Declare  $m$  a best state  $m^*$  at  $t$  if  $\Lambda_t(m^*) = \max_M(\Lambda_t(m))$  and save.

(6) (b) If  $m^*$  is not unique among the  $M$  states, then any one of  $m \in \{m^*\}$  discovered in (6) (a) may be declared the best state, and so saved.

(6) (c) Once used for an information decision, discard the previously-saved best state  $m^*$  from the previous iteration.

again, ruthless efficiency

# Viterbi Algorithm State & Best State Selection

## Step 7 – Save Identity of Branch Info

the previous state itself *could* be saved, but that's wider, i.e., more memory

(7) (a) Save the set of  $K-1$  hypothesized information bits identifying the selected transitions ~~(or, equivalently, a pair comprising each current state  $m$  across the trellis width and its selected previous state  $m$ 's oldest bit)~~ called the “identity” of the selected transition) to a memory large enough to store a sequence of such sets, from current back to oldest along the trellis, to support a desired decision delay.

(7) (b) Once used for an information decision, ~~discard the previously-oldest set of  $K-1$  hypothetical information bits (or equivalent set of identities)~~ retained in the previous iteration.

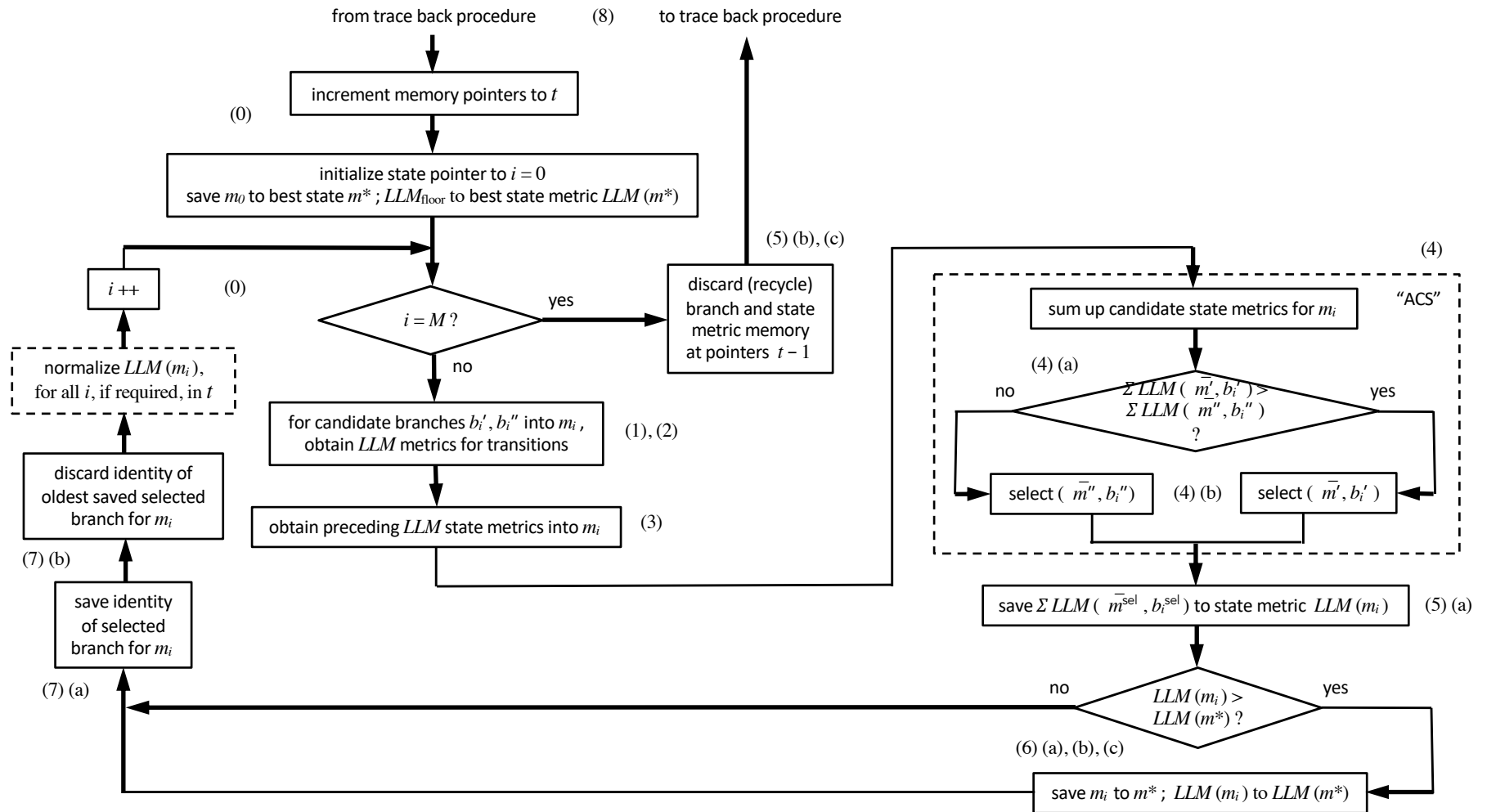
both a statement of Trace Back length, but also of Viterbi's original idea of sliding block decoding

The Algorithm, Viewed as a Flow Chart  
for (Best) State Selection, Trace Back not shown

( indexed to previous views by Step # )

# Viterbi Algorithm States & Best State Selection

## Flow Charted

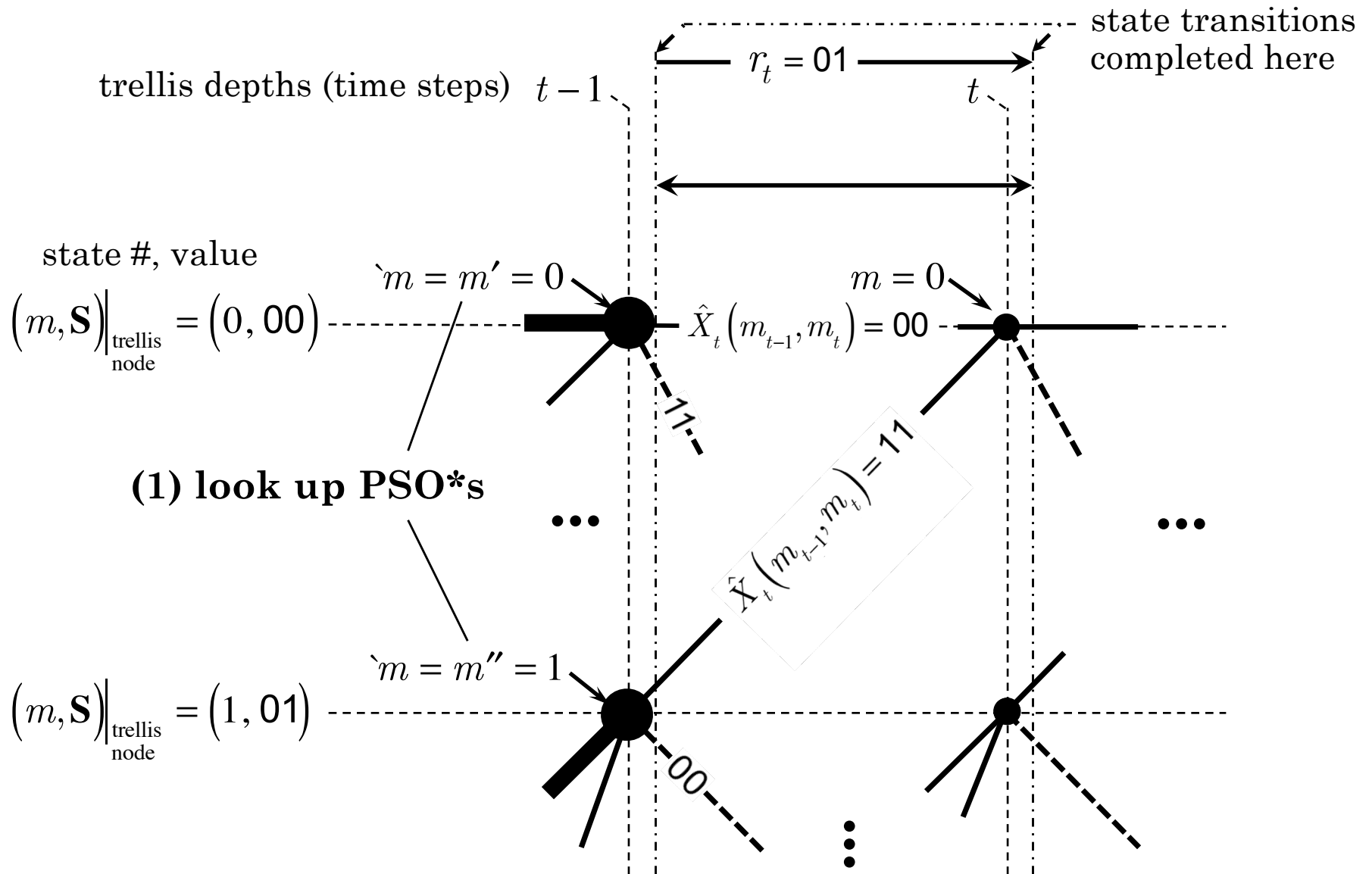


# ACS Steps, Captured “on” the Trellis at One Example State

( indexed to previous views by Step # )

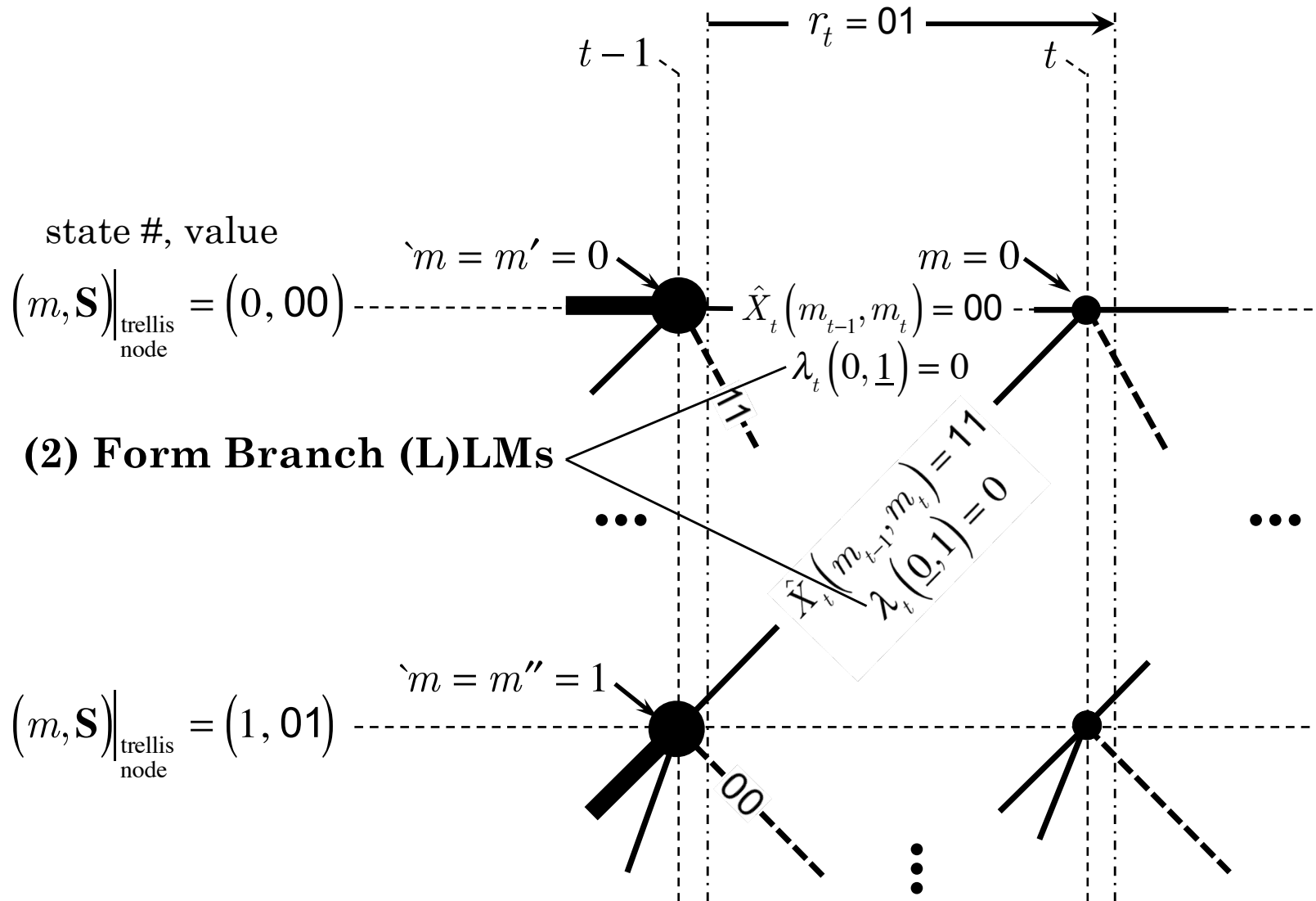


# Viterbi Algorithm (Best) State Selection, 1

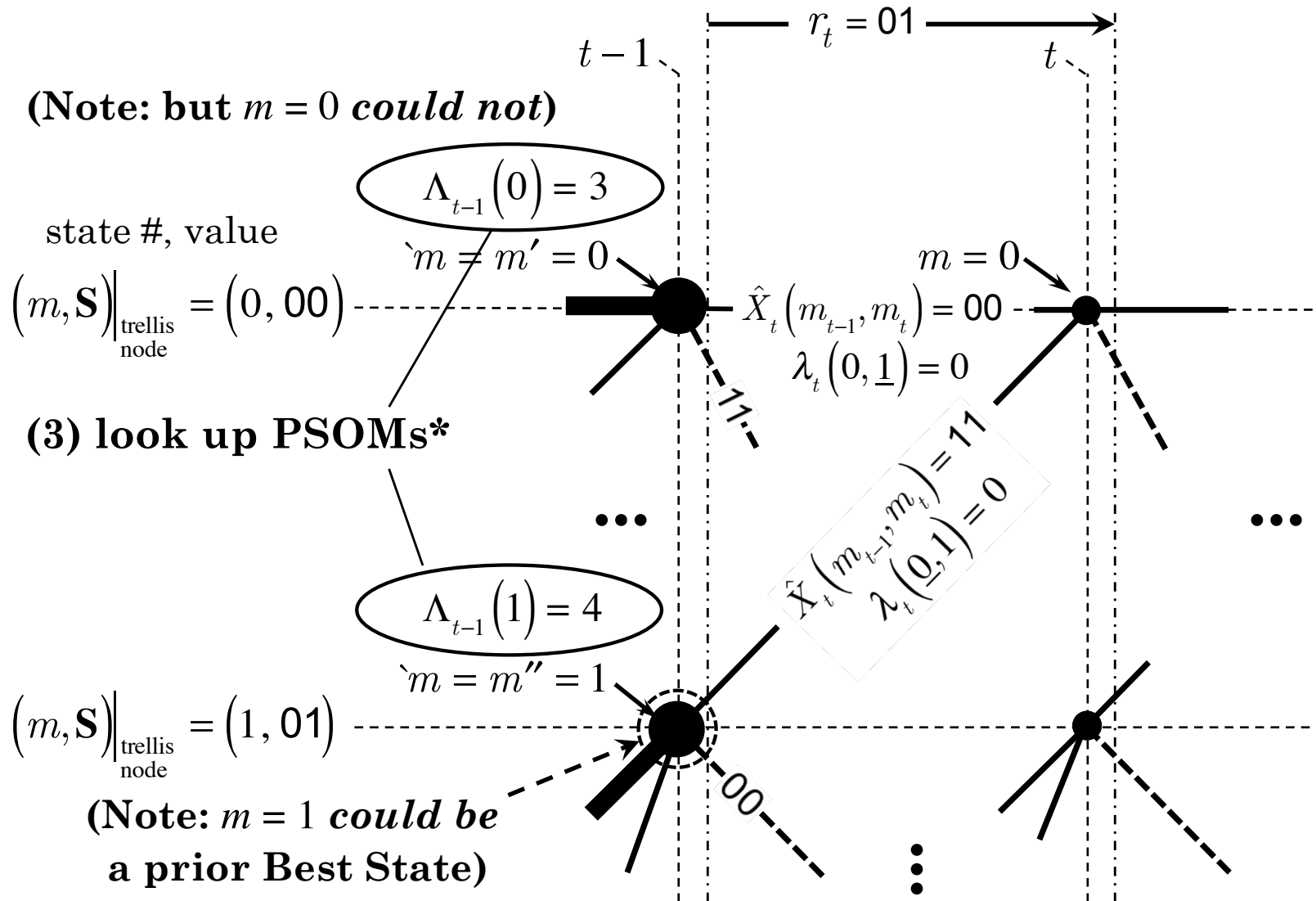


\* PSO = Preceding State-of-Origin

# Viterbi Algorithm (Best) State Selection, 2

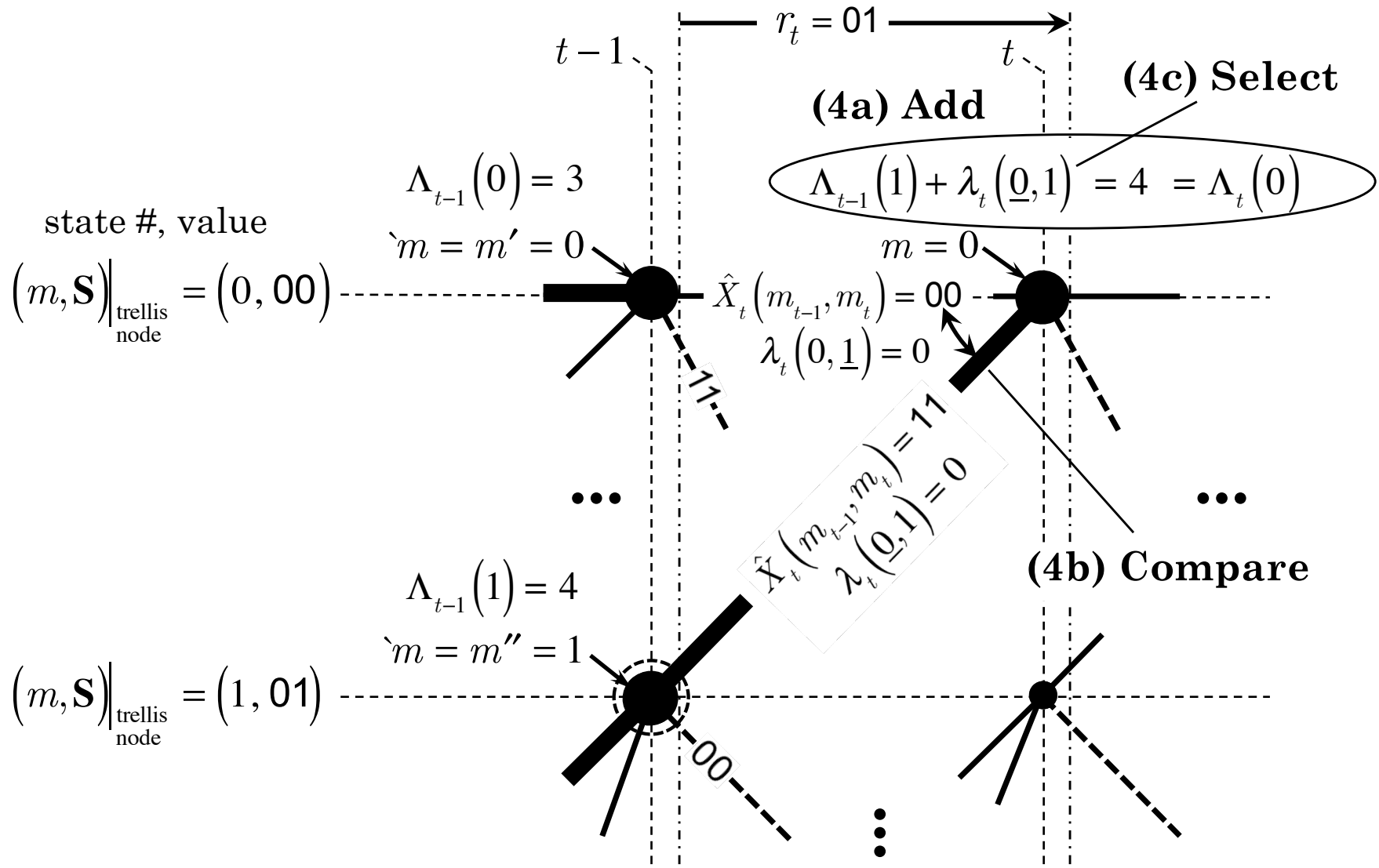


# Viterbi Algorithm (Best) State Selection, 3

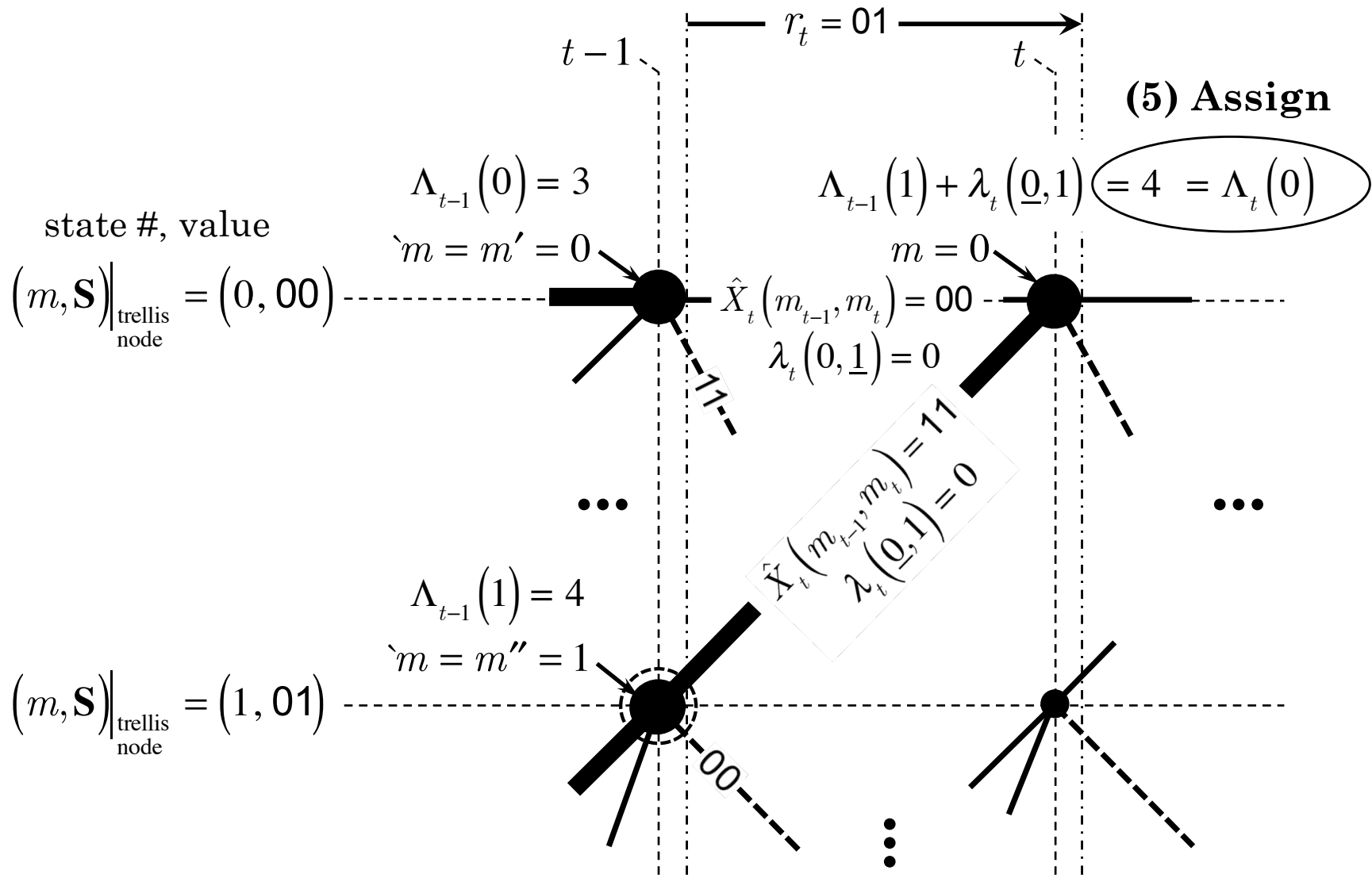


\* PSOM = Preceding State-of-Origin Metric

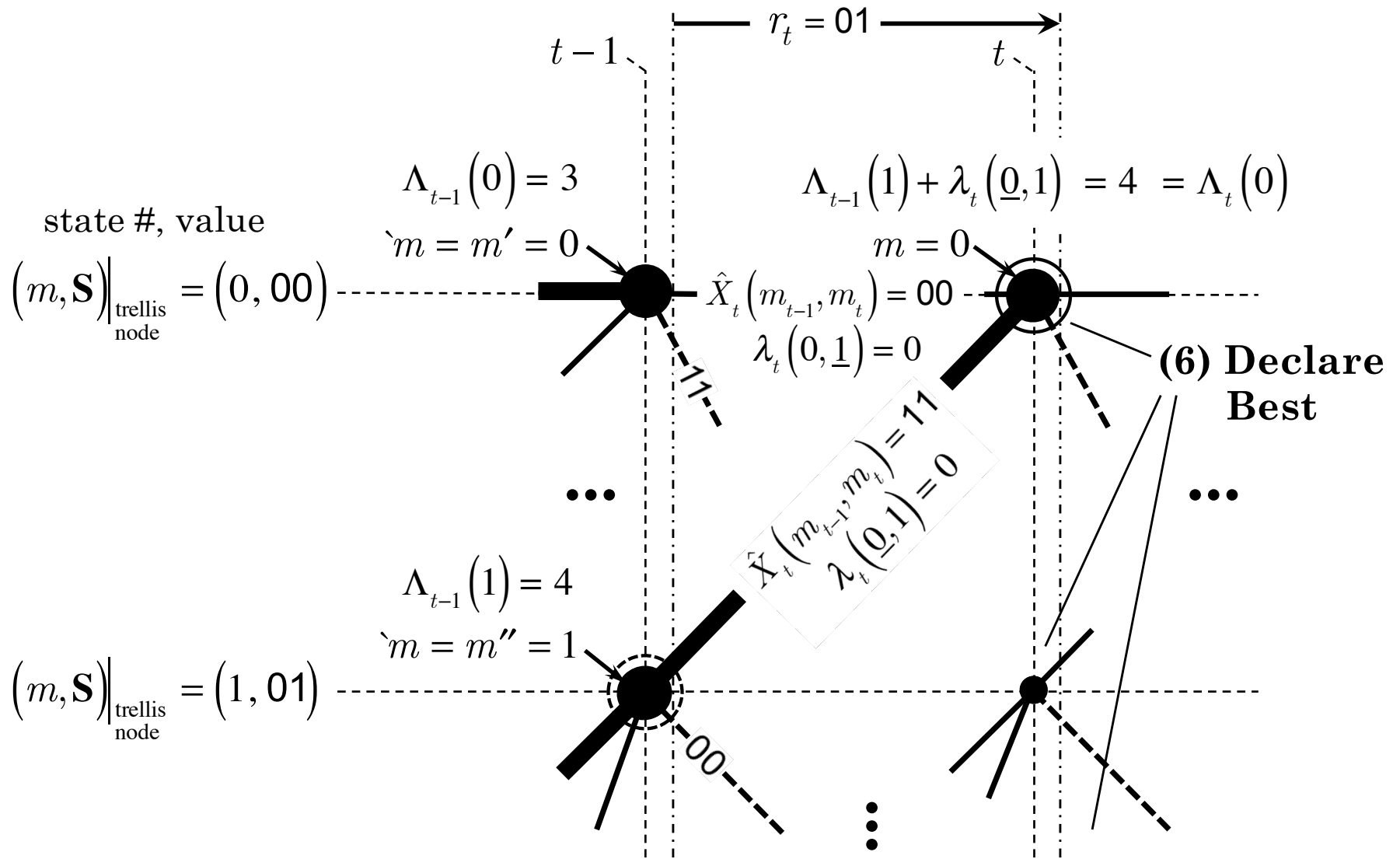
# Viterbi Algorithm (Best) State Selection, 4



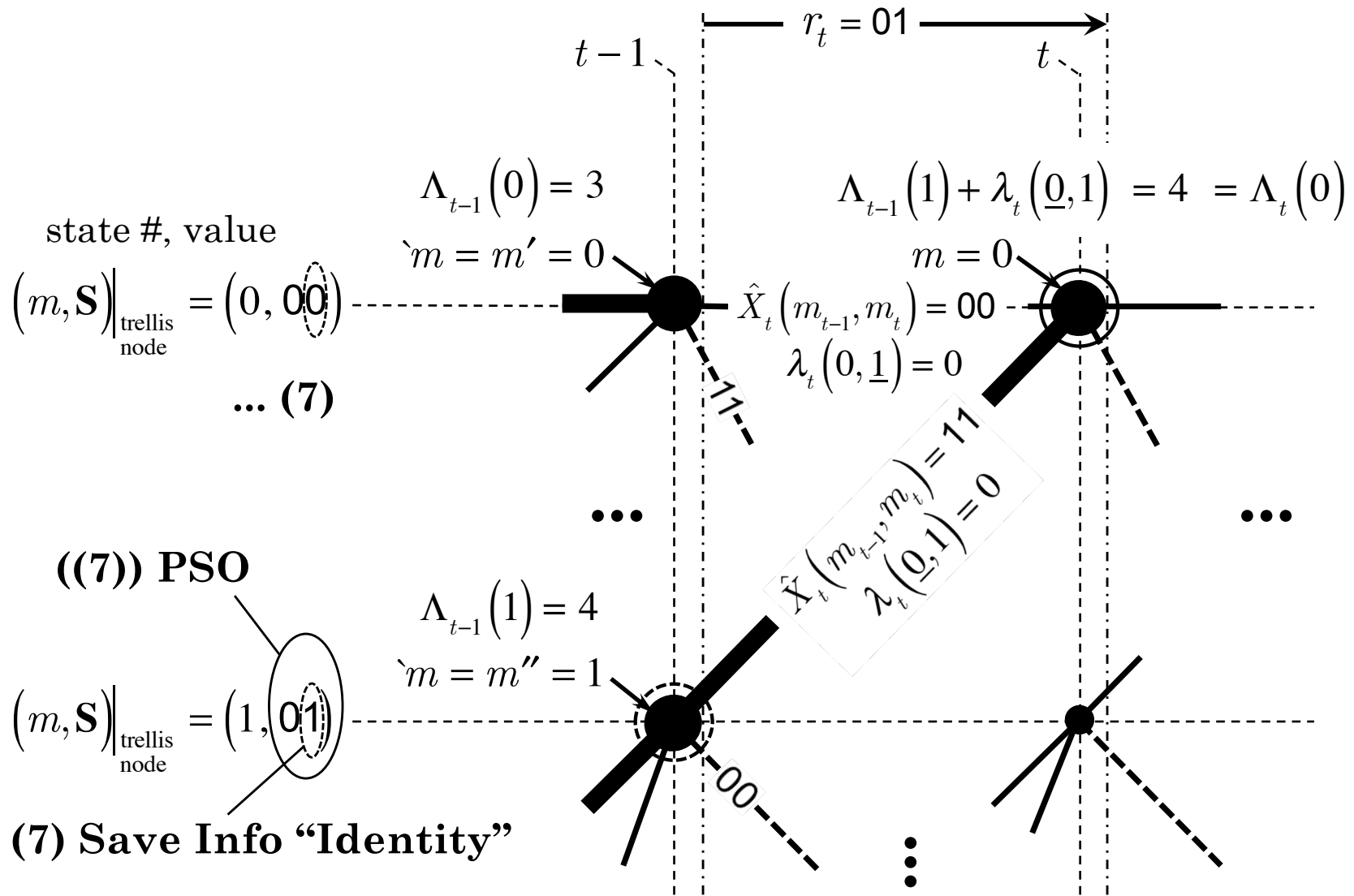
# Viterbi Algorithm (Best) State Selection, 5



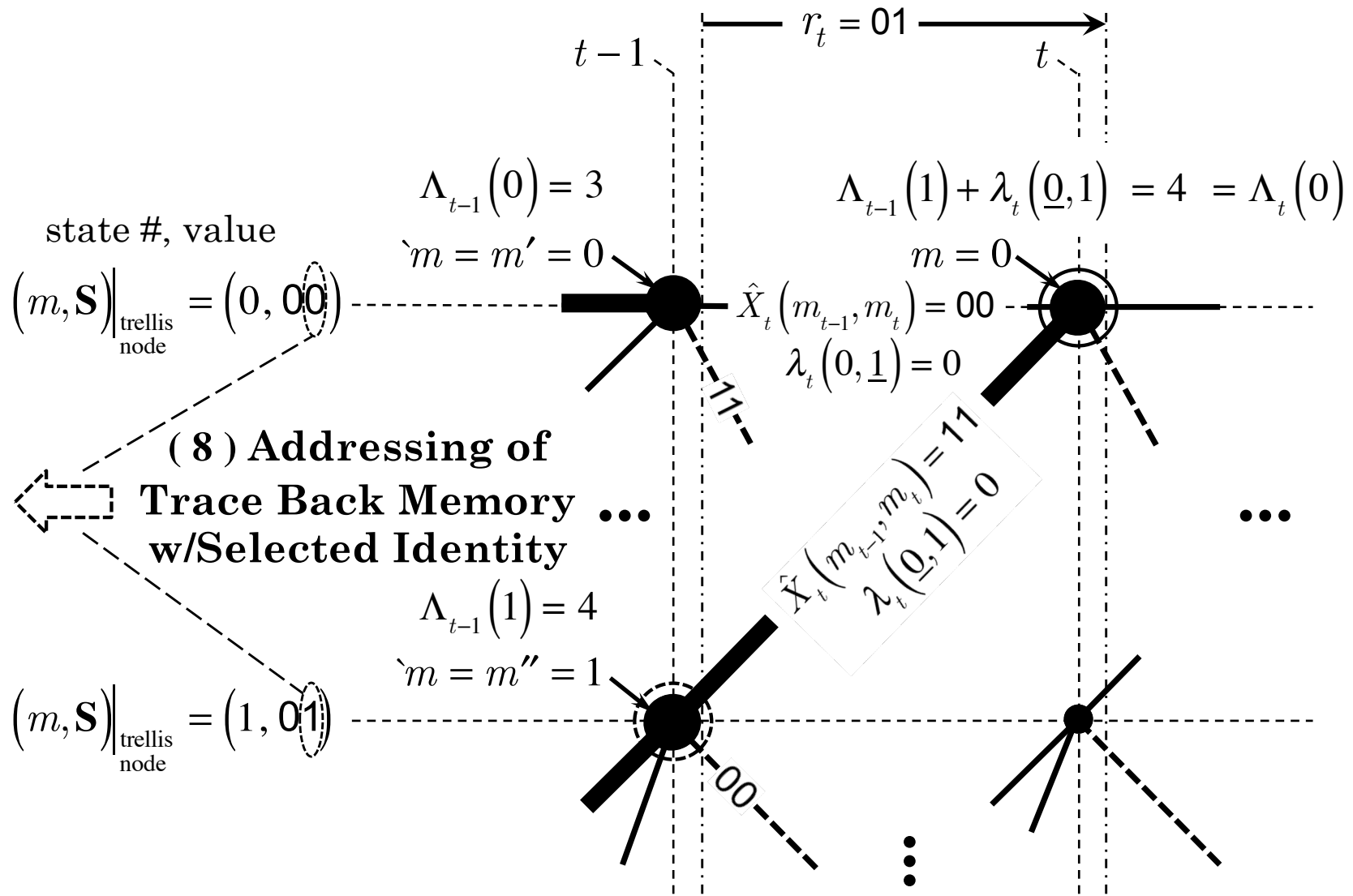
# Viterbi Algorithm (Best) State Selection, 6



# Viterbi Algorithm (Best) State Selection, 7



# Viterbi Algorithm (Best) State Selection, 8



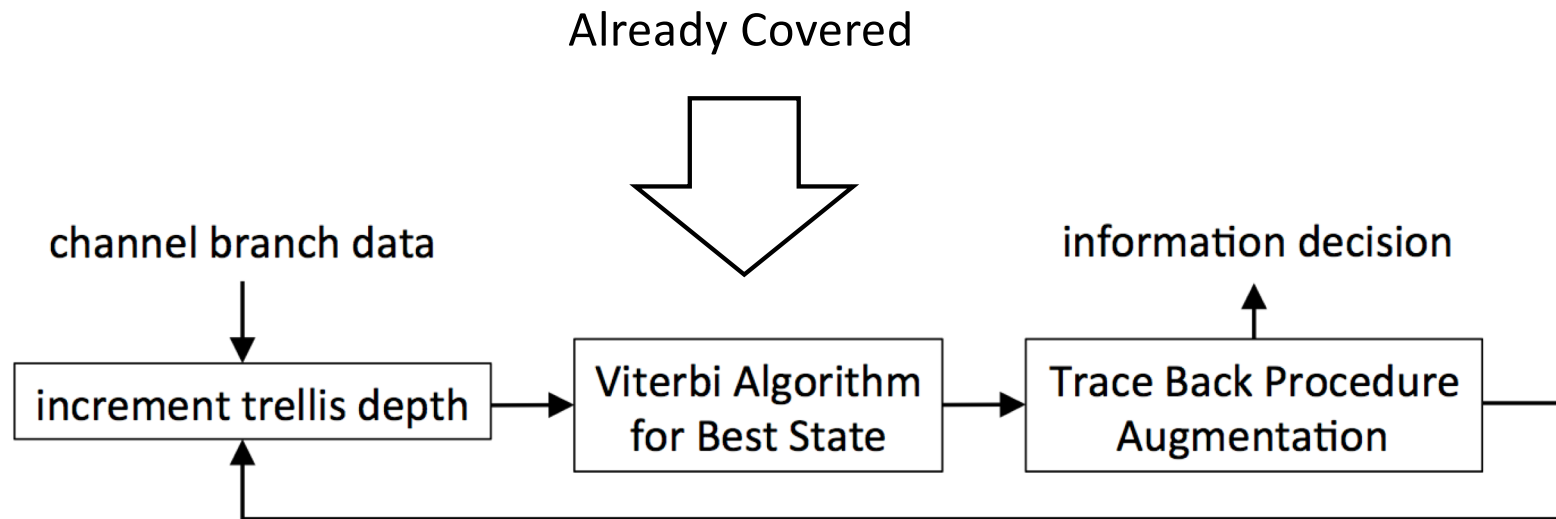


The Algorithm, Part 3 ... Trace Back

The Missing Link in Many Explanations

# Refined Viterbi Algorithm from 100,000'

## Two Relatively Separate Procedures



Trace Back will Avoid  
Decoding an Entire Block at Each Depth

# Viterbi Algorithm

## Step 8 – Decode Depends on Trace Back

The very refinement that made the VA so overwhelmingly dominant

It was never in doubt – blocks had to be extended *way* beyond  $K$

We knew this from the bounds!

(8) Execute one trellis branch worth of trace back procedure to an information decision (in lieu of decoding a block of information). Step forward in trellis and return to (1).

The procedure is now complete for trellis depth at  $t$  and the objective is achieved. The current instance, or cycle, of the algorithm has ended. The system state has been updated and recorded into a constant-sized memory via the discards. The procedure is exited in such a way that it can be repeated for the next trellis depth at  $t + 1$ .

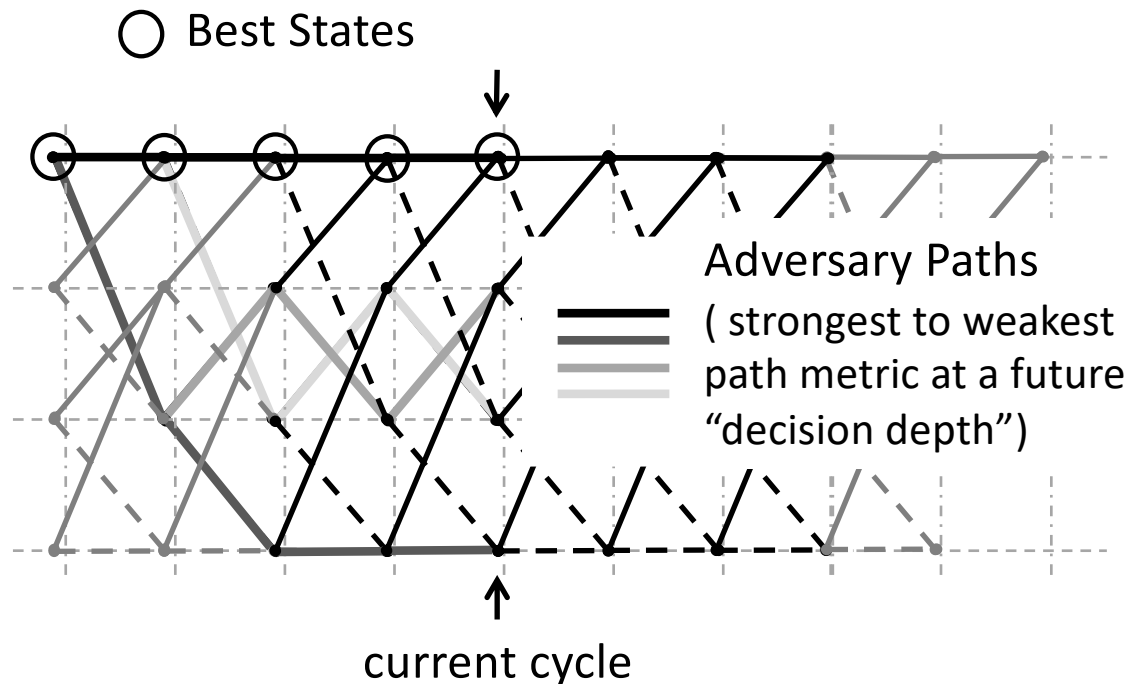
# Trellis Adversaries, 1

## Prospective Paths into Each Current State

$K = 3$  ,  $R_c = 1/2$  Example Code      Encoding All-Zeros Information

(All-Zeros is the Correct Path in this “canonical” example)

(Due to Code Linearity, All-Zeros “exemplifies” any actual codeword)

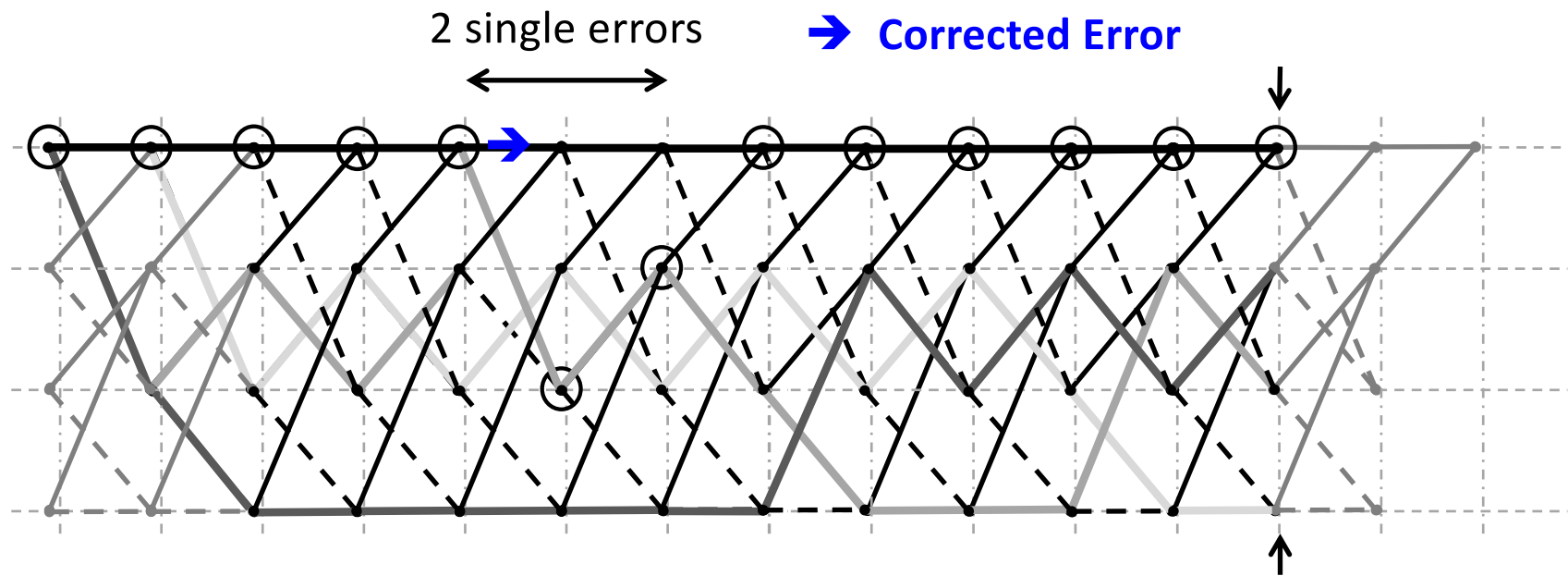


For  $R_c = 1/2$  Binary Codes  
There are always  $2^{K-1}$  Possible Codewords  
Each called an Adversary defining a legit Path to it,  
one for every State  
at any given Depth

# Trellis Adversaries, 2

## Errors & Paths from a Decode-able Channel History

$K = 3, R_c = 1/2$  Example    Error Runlength Weight  $< d_{\text{free, min}} / 2$



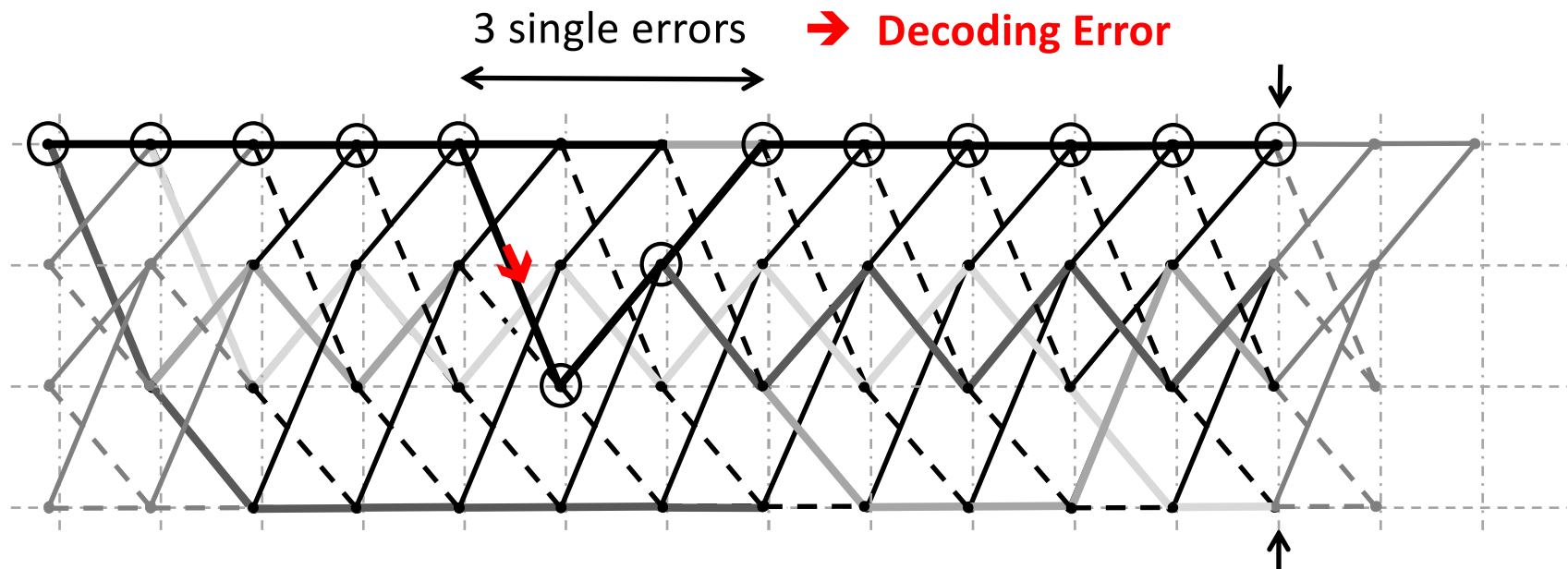
In this Example

**Any** Trace Back Length  $\geq K$  would yield a Correct Decision !

# Trellis Adversaries, 3

## Errors & Paths from an Undecode-able Channel History

$K = 3, R_c = 1/2$  Example Error Runlength Weight  $\geq d_{\text{free, min}} / 2$



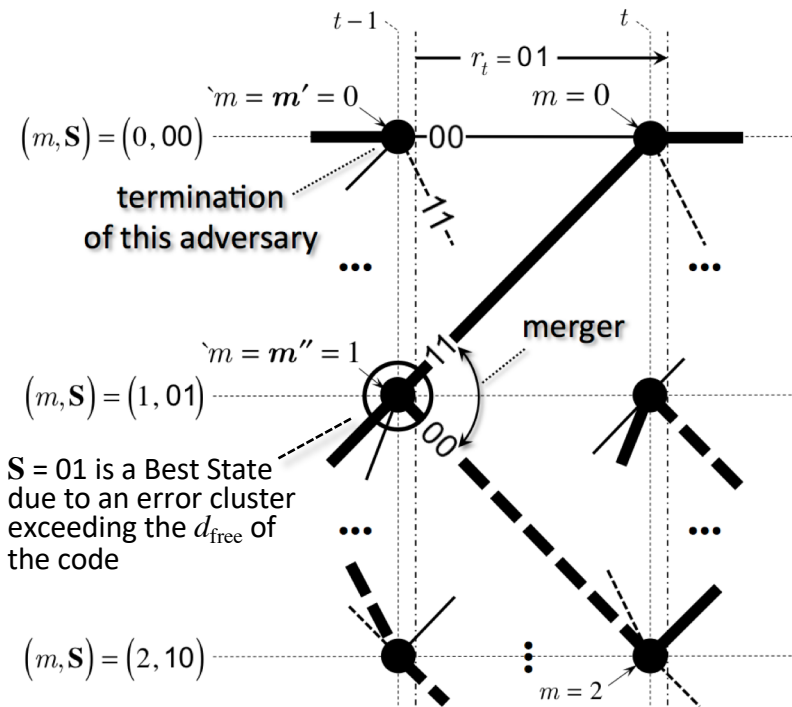
In this Example

**No** Trace Back Length would yield a Correct Decision !  
**Any** Code and Decoder can make a Decoding Error

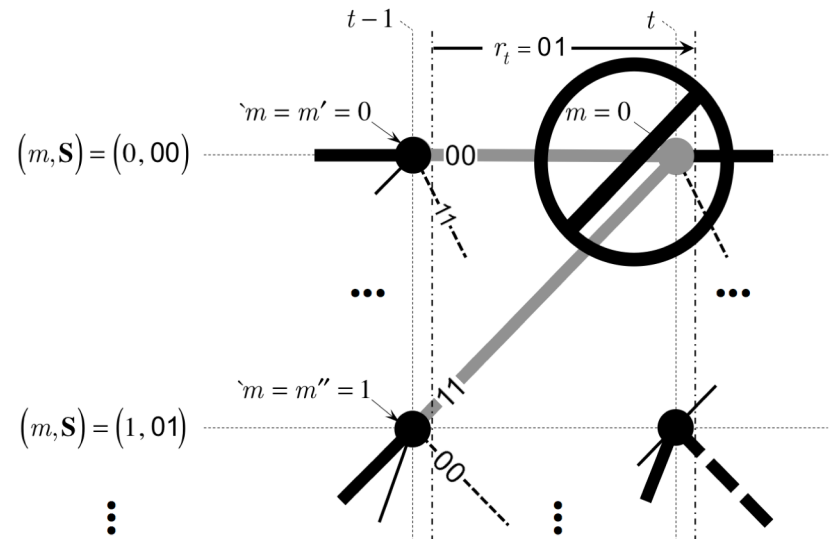
# Trellis Adversaries, 3

## A Fatal Merger on the Decoder Trellis

Mergers are “Backward-Looking” along Branch Segments during Trace Back Viterbi called them “Diverging” Paths in his Paper, in the Forward Time Sense



No Such Thing as a Forward Merger !



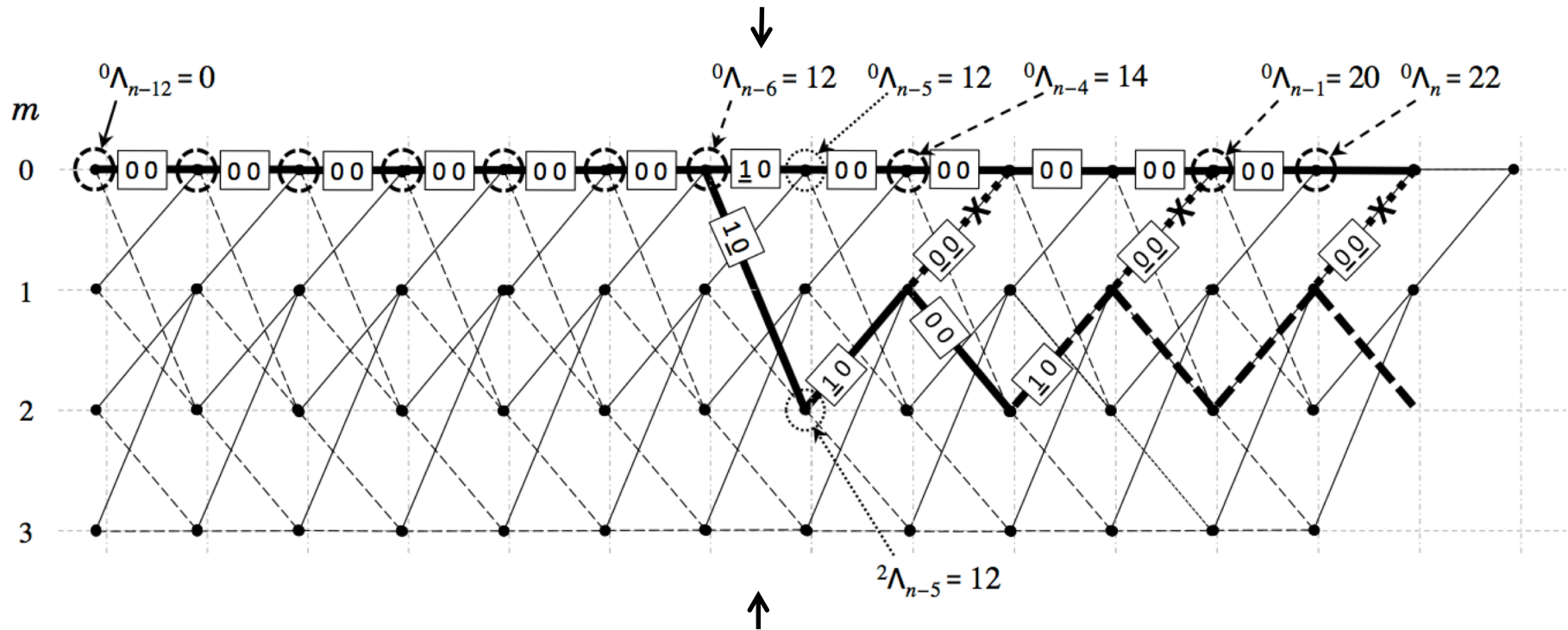
For All-Zeros Codeword Sent, This Merger Can Only Happen due to an Error Cluster !  
It WILL Result in Decoded Errors – Since the 11 Path from State 00 is Also Cut Off !

# Trellis Adversaries, 4

## Path Metrics Along the Two Best Adversaries

$K = 3, R_c = 1/2$  Example When Error Runlength Weight  $\leq d_{\text{free, min}} / 2$

All Zeros Sent, One Error (in Runlength)



Trace Back of Just One Branch is Enough !

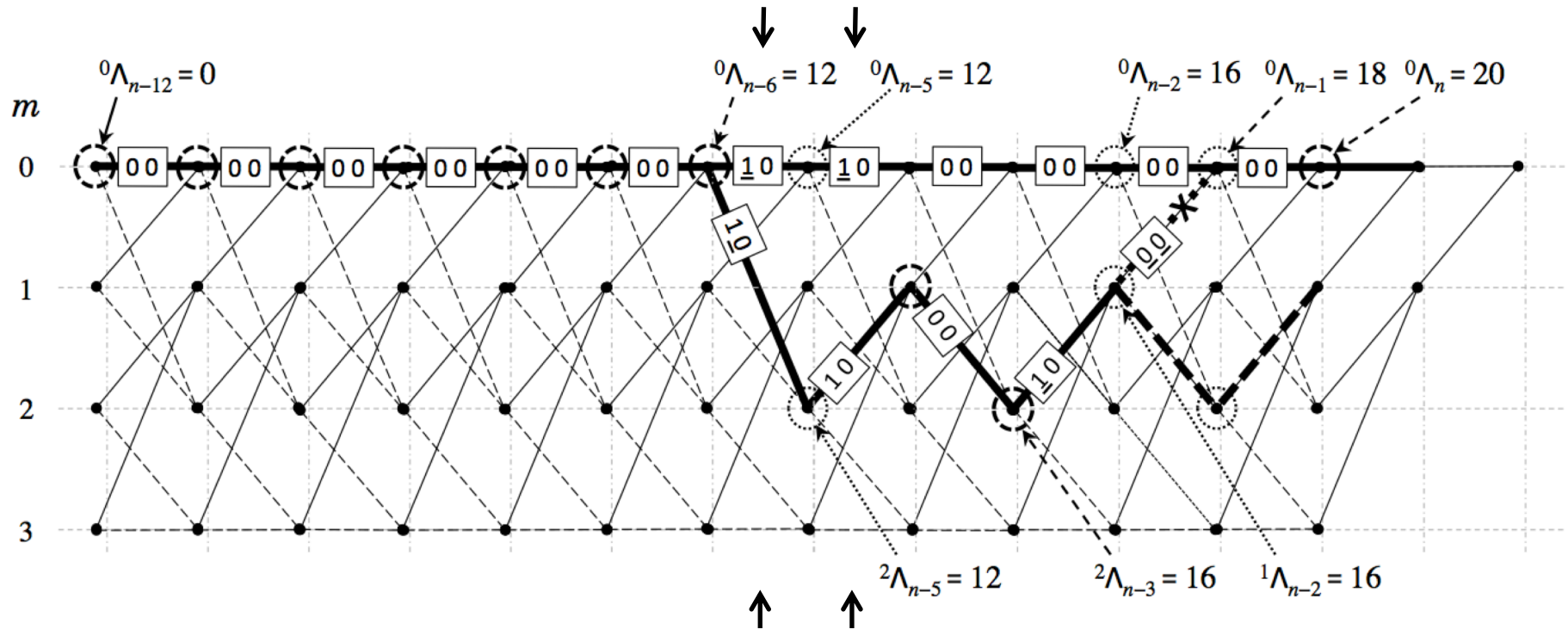


# Trellis Adversaries, 5

## Path Metrics Along the Two Best Adversaries

$K = 3, R_c = 1/2$  Example When Error Runlength Weight  $\leq d_{\text{free, min}} / 2$

All Zeros Sent, Two Errors (in Runlength)



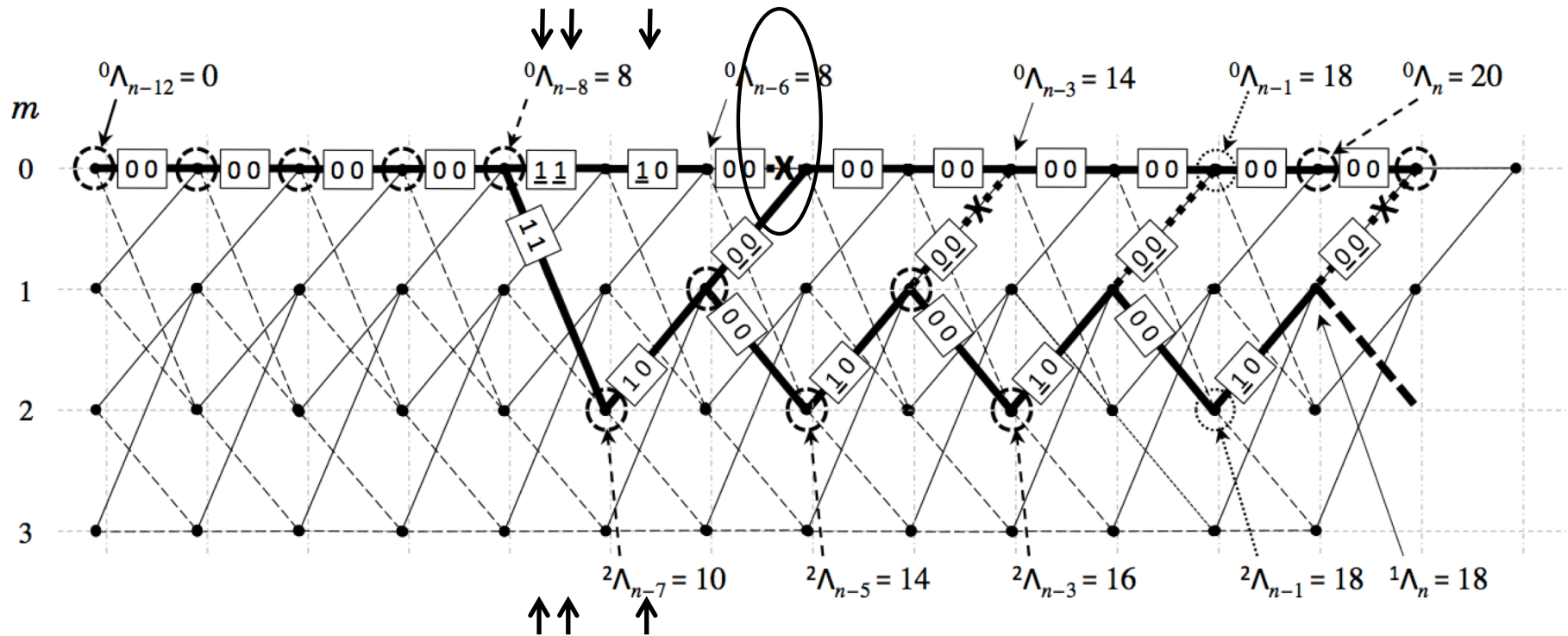
Longer Trace Back is Required (or Else an Error) !

# Trellis Adversaries, 6

## Path Metrics Along the Two Best Adversaries

$K = 3, R_c = 1/2$  Example When Error Runlength Weight  $> d_{\text{free, min}} / 2$

All Zeros Sent, Three Errors (in Runlength)



No Length of Trace Back is Sufficient to Avoid Error !

# Decoding in Viterbi's 1967 Paper

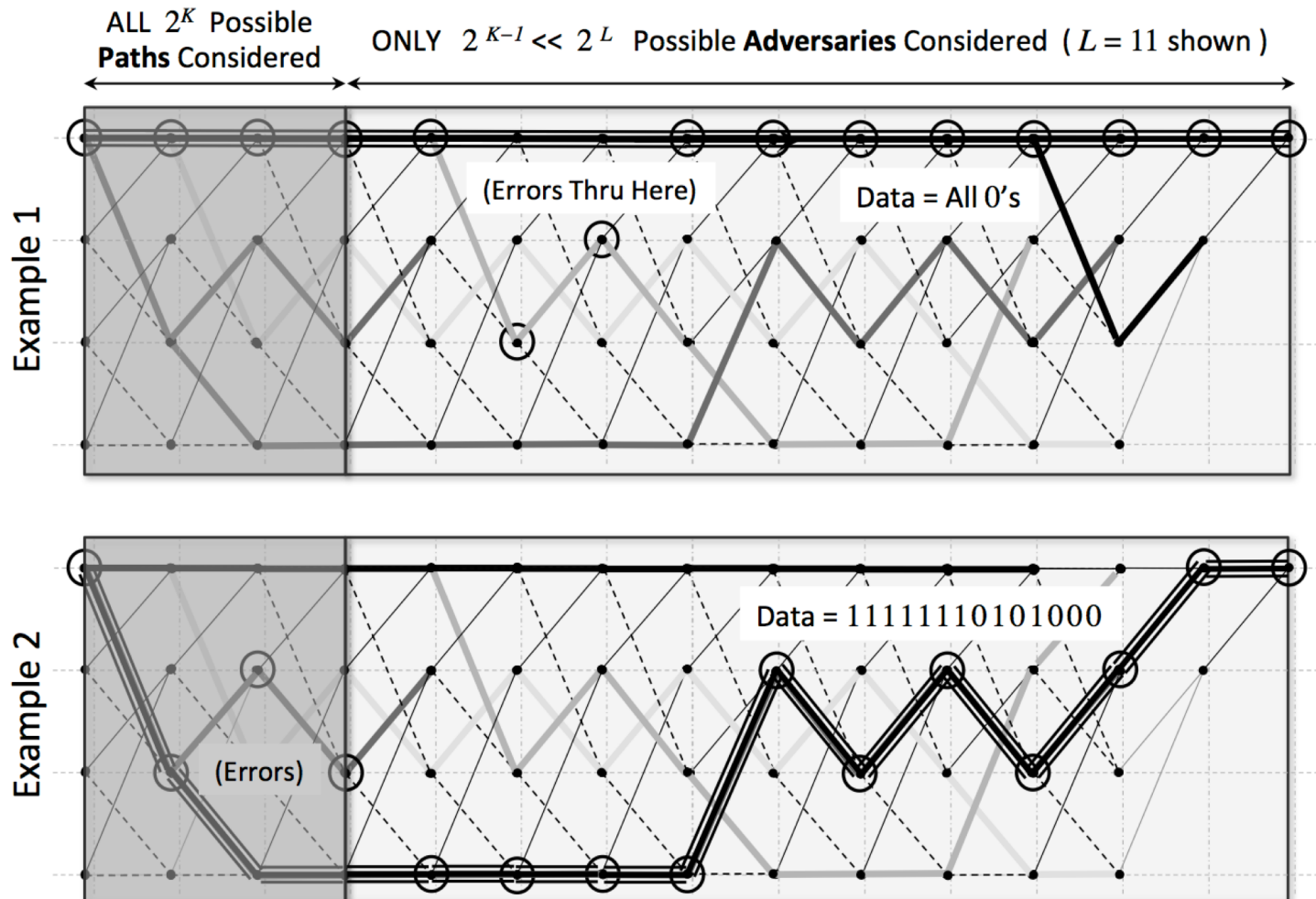
## For $q = 2$ , A Tale of Two\* Adversaries

The **Nonsequential Algorithm** (Part IV) Describes State Metrics *and Decoding* !  
*State Metrics become Path Metrics* over any Block beyond  $L = K$

2 Examples for  
 $K = 3, R_c = 1/n$

Limiting the Algorithm's Consideration to  $2^{K-1}$  Alternative Choices *Everywhere* (Beyond the first  $K$  "Branches") was the **Blockbuster Breakthrough** of the VA

\* always being the "best" and the "next best"

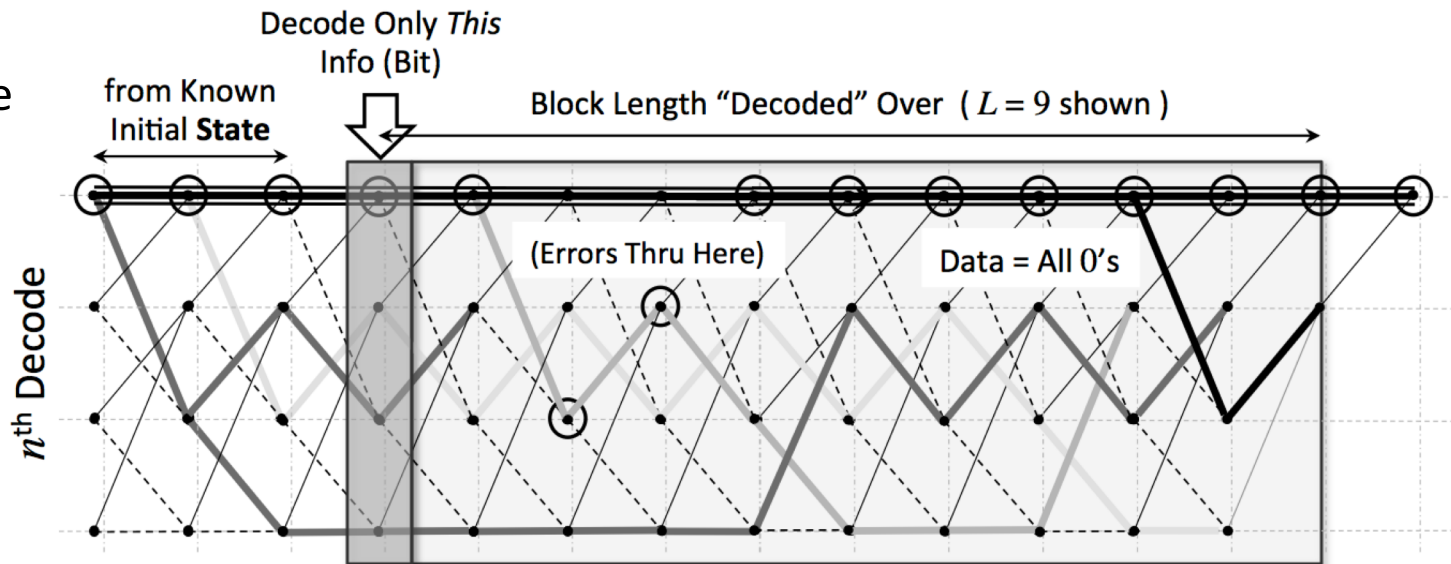


# Decoding Beyond Viterbi's 1967 Paper, 1

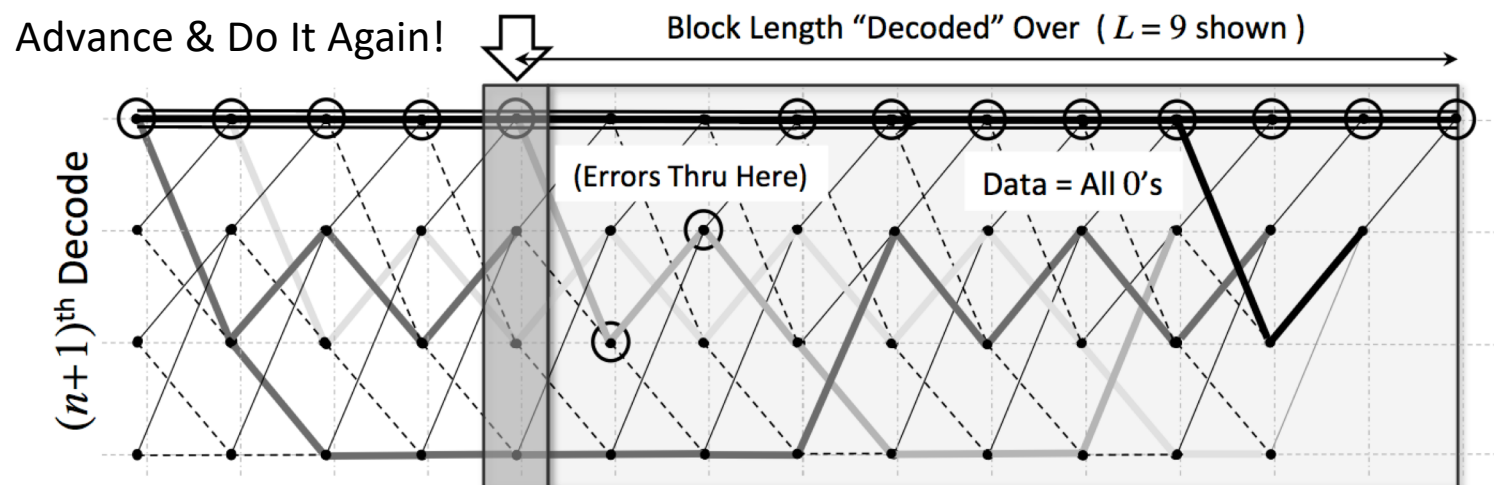
## A Refined Viterbi Algorithm by *Trace Back Decoding*

"Start-Up" Example  
 $K = 3, R_c = 1/n$   
 Trace Back  $3K$

VA's **Blockbuster Breakthrough**  
 Still Prevails  
 BUT Each Block now Overlaps the Previous



Buys Advantage of Many Blocks and *the* Most Likely Decision Decode of Each Oldest Branch, One by One

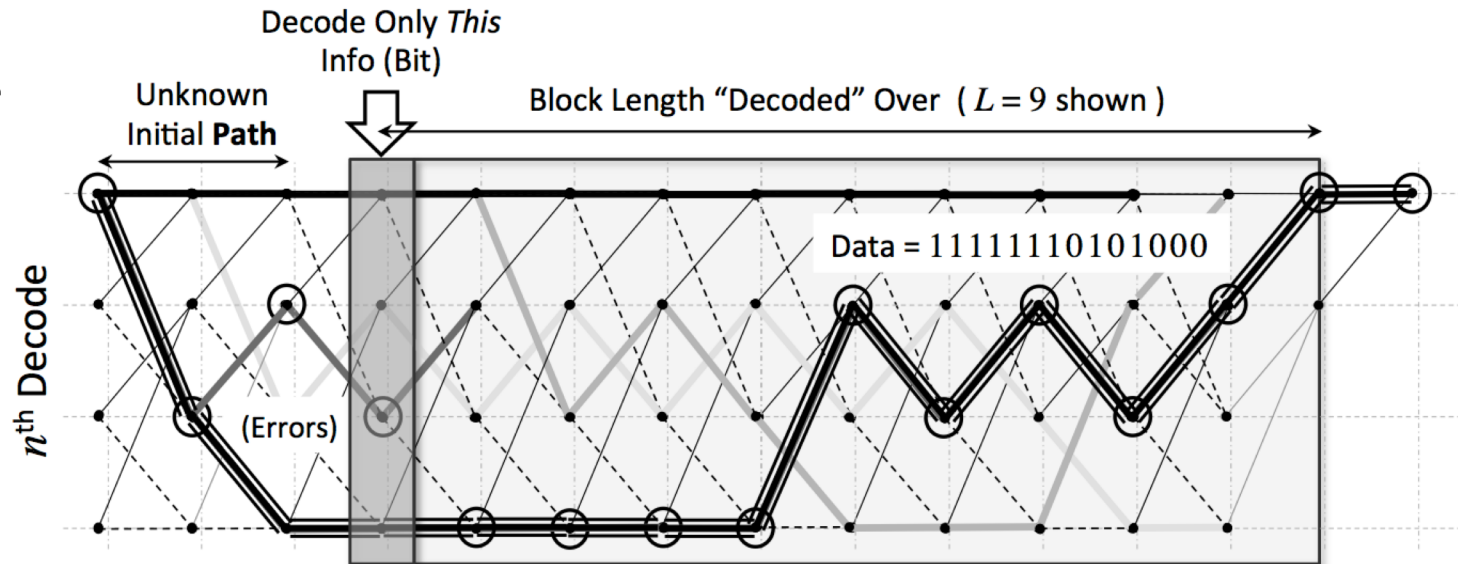


# Decoding Beyond Viterbi's 1967 Paper, 2

## A Trace Back Decoding Example for Longer Blocks

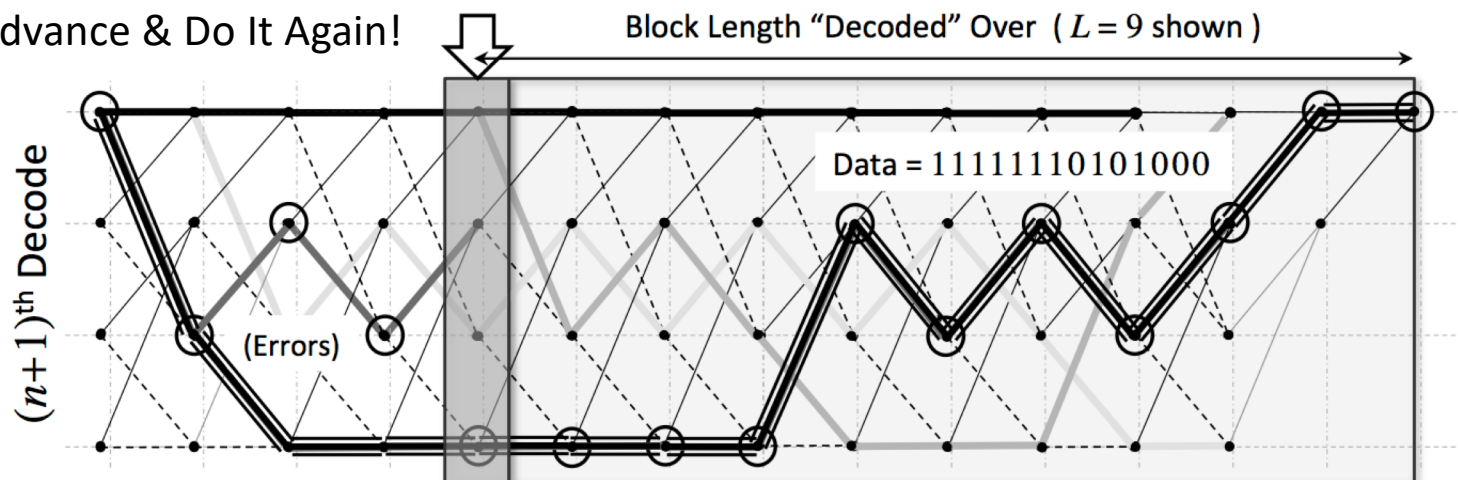
"Jump-In" Example  
 $K = 3, R_c = 1/n$   
 Trace Back  $3K$

It Doesn't Matter  
 Where the  
 Algorithm  
 Begins!



Advance & Do It Again!

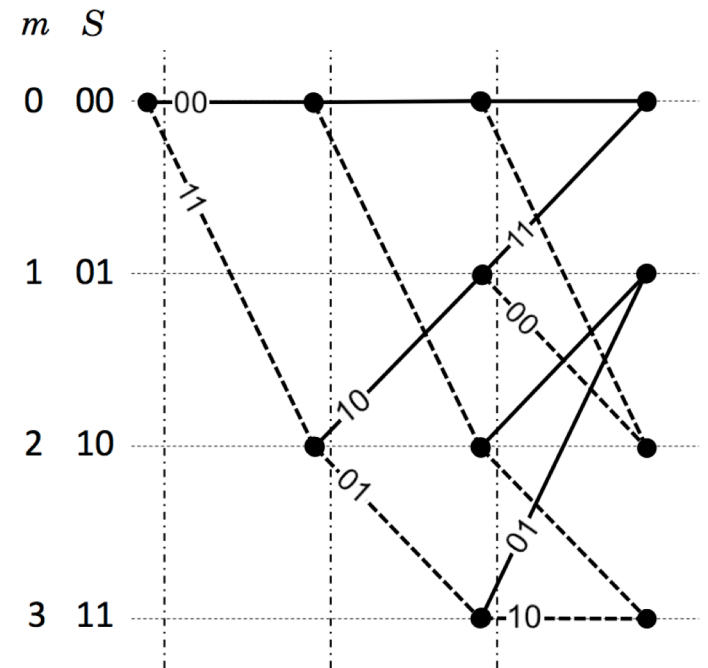
Notice that  
 $L = K+1$   
*Just Barely*  
 Works, i.e. Trace  
 Back  $1K$   
 Is Not Robust!



# Keys to Trace Back Length, 1

## Two Factors Seldom Mentioned Together

- Factor 1: Behavior of the LLM branch  $\lambda_t$ 
  - “double errors” have a big penalty relative to single errors
- Factor 2: Mergers
  - where adversaries share a common “root”  $\Lambda_t(m)$
- Recall our canonical trellis
  - binary  $K = 3, R_c = 1/2$
  - just to demonstrate branches
- Typical  $K = 7, R_c = 1/2$  3SD case
  - channel  $E_b/N_0 = 5$  dB
    - yielding output  $P_b = 10^{-6}$
  - corresponds to  $p = 3 \times 10^{-2}$



# Keys to Trace Back Length, 2

## Single Error (on a Branch) Case

- Comparing transitions at Mergers
  - we reverse the state transition nomenclature
- Consider

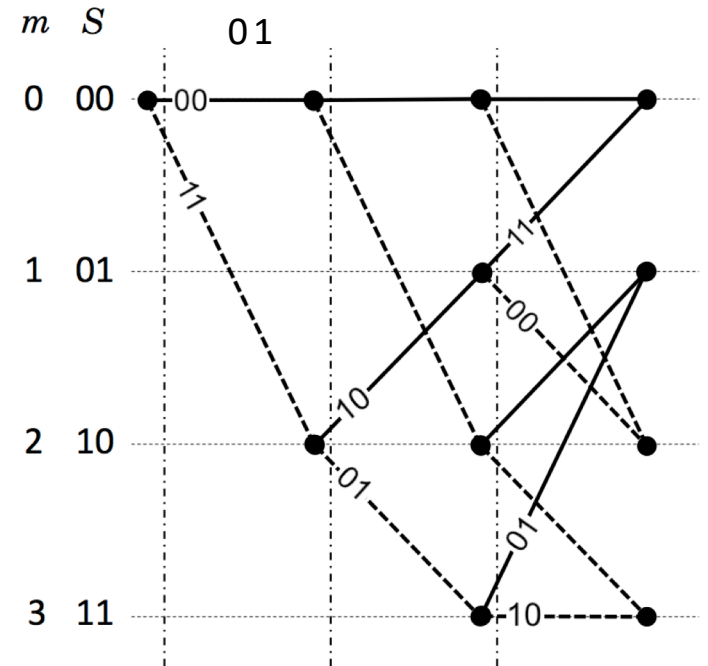
$$(m, m') = (00, 00) \quad (m, m'') = (00, 10) \quad \textcircled{Y_t = 0 \ 1} \quad \hat{X}_t(00, 00) = 00 \quad \hat{X}_t(00, 10) = 11$$

$$\lambda_t(00, 00) = \frac{\Pr\{01 \mid 00\}}{\Pr\{01 \mid 11\}} = \frac{(1-p)p}{p(1-p)} = 1$$

$$1 = \frac{p(1-p)}{(1-p)p} = \frac{\Pr\{01 \mid 11\}}{\Pr\{01 \mid 00\}} = \lambda_t(00, 10)$$

$$\log \lambda_t(00, 00) = \log \lambda_t(00, 10) = 0$$

- At any  $p \rightarrow \log_{10}(\dots) = 0$



# Keys to Trace Back Length, 3 Double Error (on a Branch) Case

- Now Consider

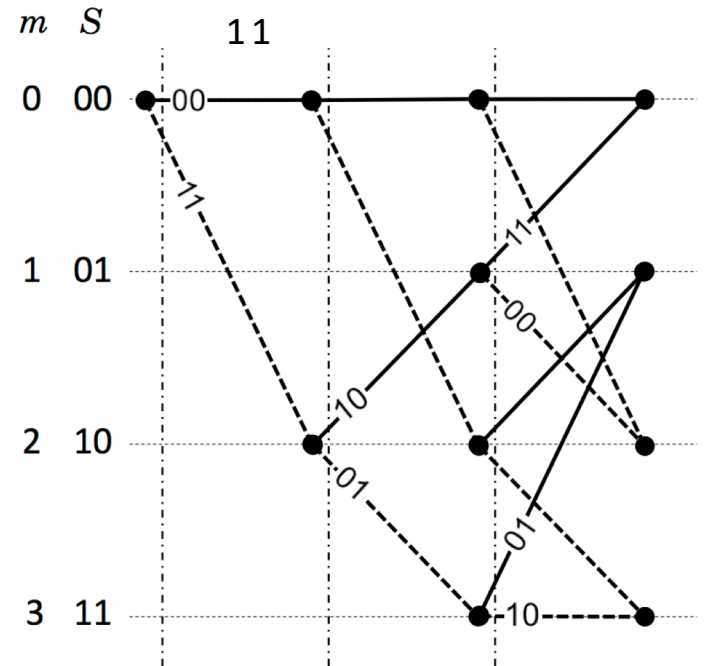
$$(m, m') = (00, 00) \quad (m, m'') = (00, 10) \quad \textcircled{Y_t = 11} \quad \hat{X}_t(00, 00) = 00 \quad \hat{X}_t(00, 10) = 11$$

$$\lambda_t(00, 00) = \frac{\Pr\{11 | 00\}}{\Pr\{11 | 11\}} = \frac{p^2}{(1-p)^2}$$

$$\frac{(1-p)^2}{p^2} = \frac{\Pr\{11 | 11\}}{\Pr\{11 | 00\}} = \lambda_t(00, 10)$$

$$\log \lambda_t(00, 00) = - \log \lambda_t(00, 10)$$

- $p = 3 \times 10^{-2} \rightarrow \log_{10}(\dots) \sim 3 \quad (-3)$





# A Key to Trace Back Length, 4

## Merger Double Error Impact

- When double errors strike at a Merger
  - A large deficit opens up two adversaries
  - It can take longer than  $1K$  to resolve even for “isolated doubles”
- A double error can be part of a “error cluster”
  - Clusters are characterized by a “runlength”
  - Consider runlengths as a multiple of  $K$
  - Added burden of *nearby single errors* opens up the deficit wider
  - Eventually, it can take  $nK$ ,  $n > 2, 3, 4$  to resolve for larger  $K$
- Sure, these clusters are exceptionally low probability
  - On a discrete *memoryless* channel
  - But they must either be resolved reliably
  - Or else be lower in event probability than the desired output  $P_b$

# Trace Back Memory (TBM)

## There are Two Basic Schemes

- Factors Common to Either
  - TBM *must be* Constructed Going Forward
  - Total Bits Constant Regardless of Info Rate
- High Info Rate Scheme
  - Parallel TBM Accessed in One Cycle
  - Expensive, Unavoidably Large Footprint
- Lower Info Rate Scheme
  - Serial TBM Access in  $m K$  Cycles
  - Much Cheaper, Potentially Tiny Footprint

### HS Parallel

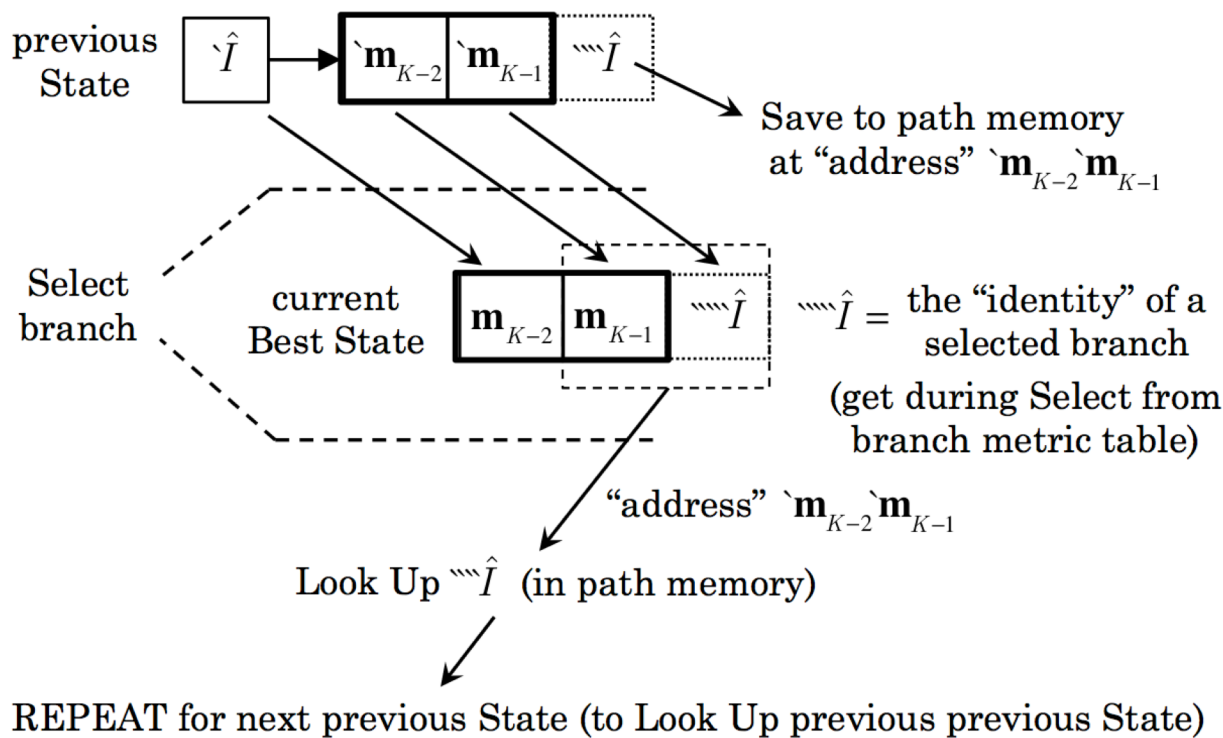
4 x ( $2^{K-1}$  x 8) RAM chips  
doubled for swapping

### LR Series

1 x ( $2^{K+3}$  x 1) RAM chip  
doubled for swapping

# Chain Back

- J. Heller’s great idea, mainly for “Series” VA Decoding
- You get Trace Back Memory addressing almost for free!
  - Exploiting “identities” (VA State Selection Step 7) already saved



In Series VA form, do this procedure at every state  $m$

It runs in parallel without any added time overhead

For  $K = 7$ , there are enough states to Trace Back 64 trellis times, but 35 suffices

# Implementing a Practical Decoder In Easy Steps onto a Block Diagram

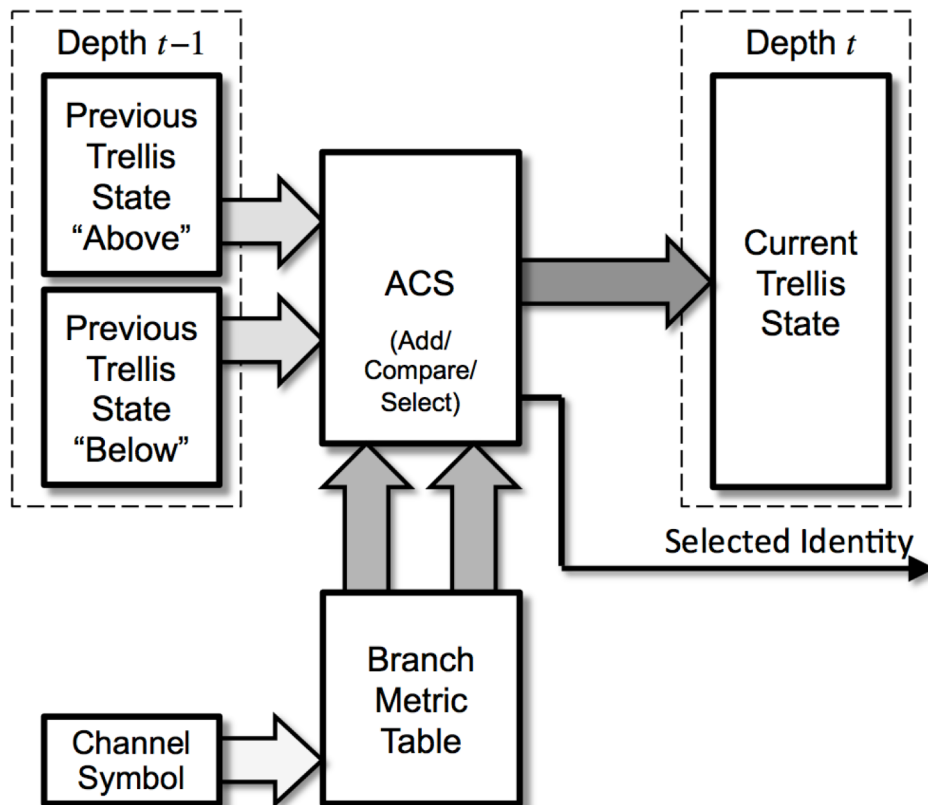
# The Viterbi Decoder, In 10 Steps

- Build Up a Complete Decoder as a Block Diagram
  - In Increasing Complexity as Each Prior Build Implies the Next Blocks
- Start from ACS Math/Logic, the “Core” VA Operation
  - Viterbi Considered ACS the VA’s Essential Compact Expression
- ACS with its Surrounding I/O Blocks Will Assume that
  - Decoder is Synchronized to Channel Symbols
  - Channel Symbols are Translated, State-by-State, into Branch Metrics
  - This latter shown explicitly as part of ACS
- The “State Clock” is Not Shown for Clarity
  - State Clock is  $2^{K-1}$  times rate of Step (Forward in Trellis) Clock
  - State Clock distributes to State Address (counter) and other logic

# Building Up a Viterbi Decoder, 1

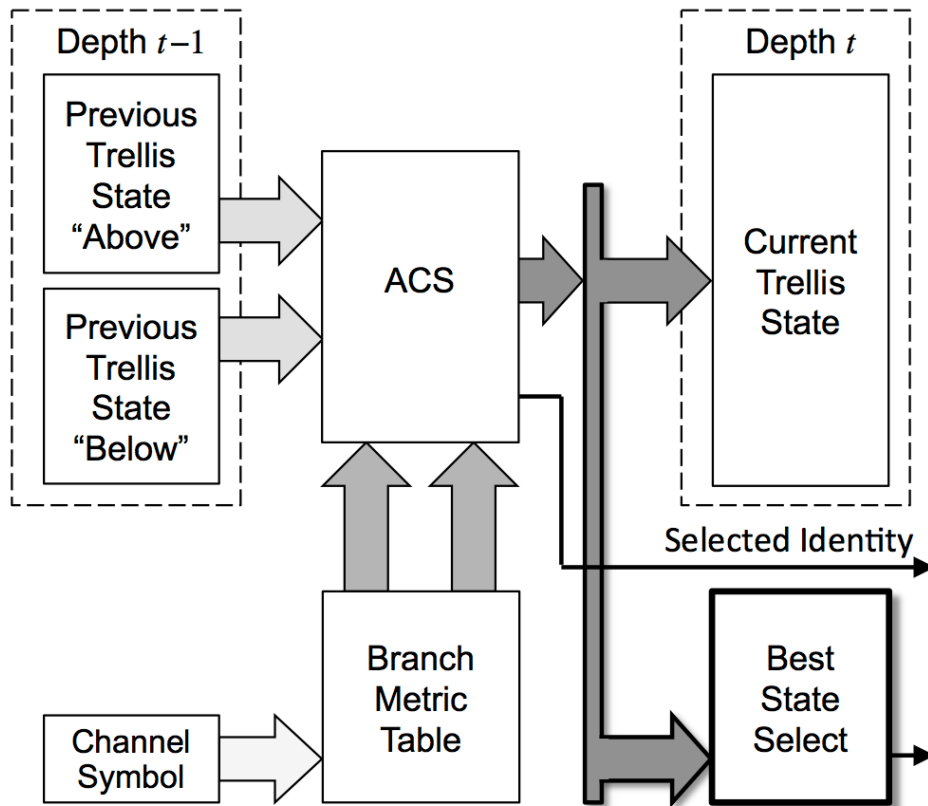
## Essential Parts of the “Core” Operation

The Decoder is assumed to be “in Sync” with the Channel  
This is easily accomplished in any of several ways, not our focus here



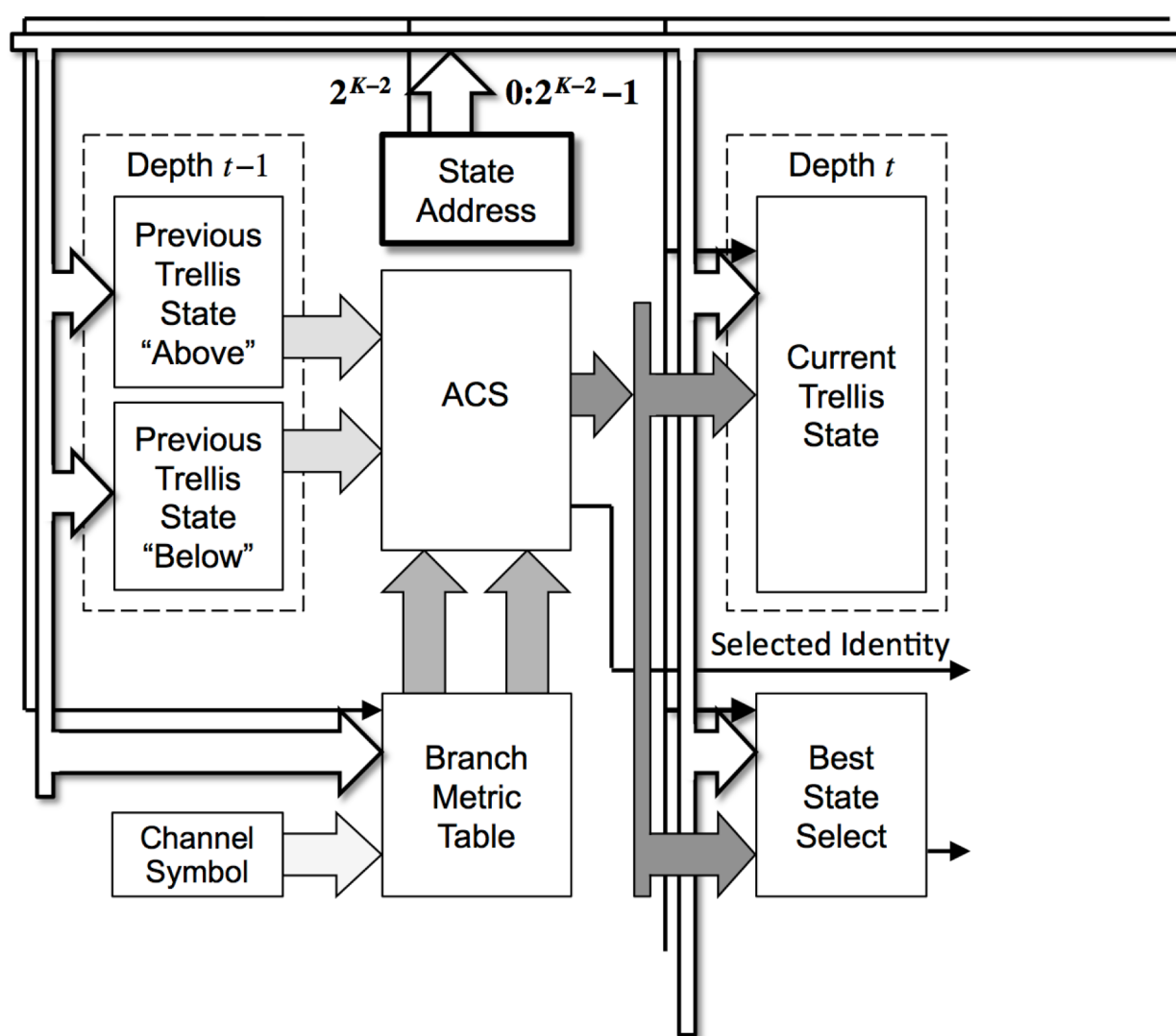
# Building Up a Viterbi Decoder, 2

## Sorting for Best State at Each Trellis Depth



# Building Up a Viterbi Decoder, 3

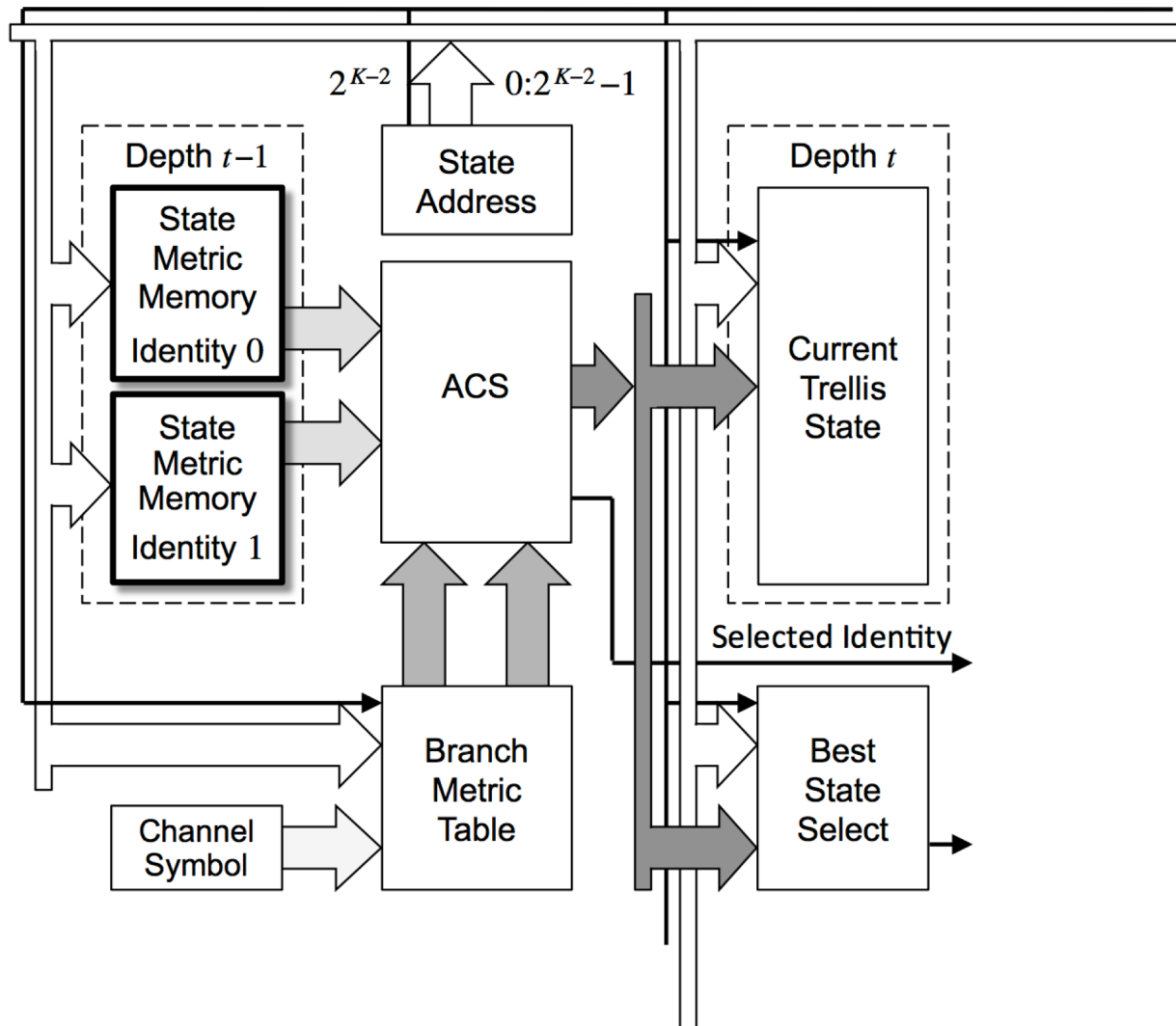
Providing successive State Addresses to Core Operation





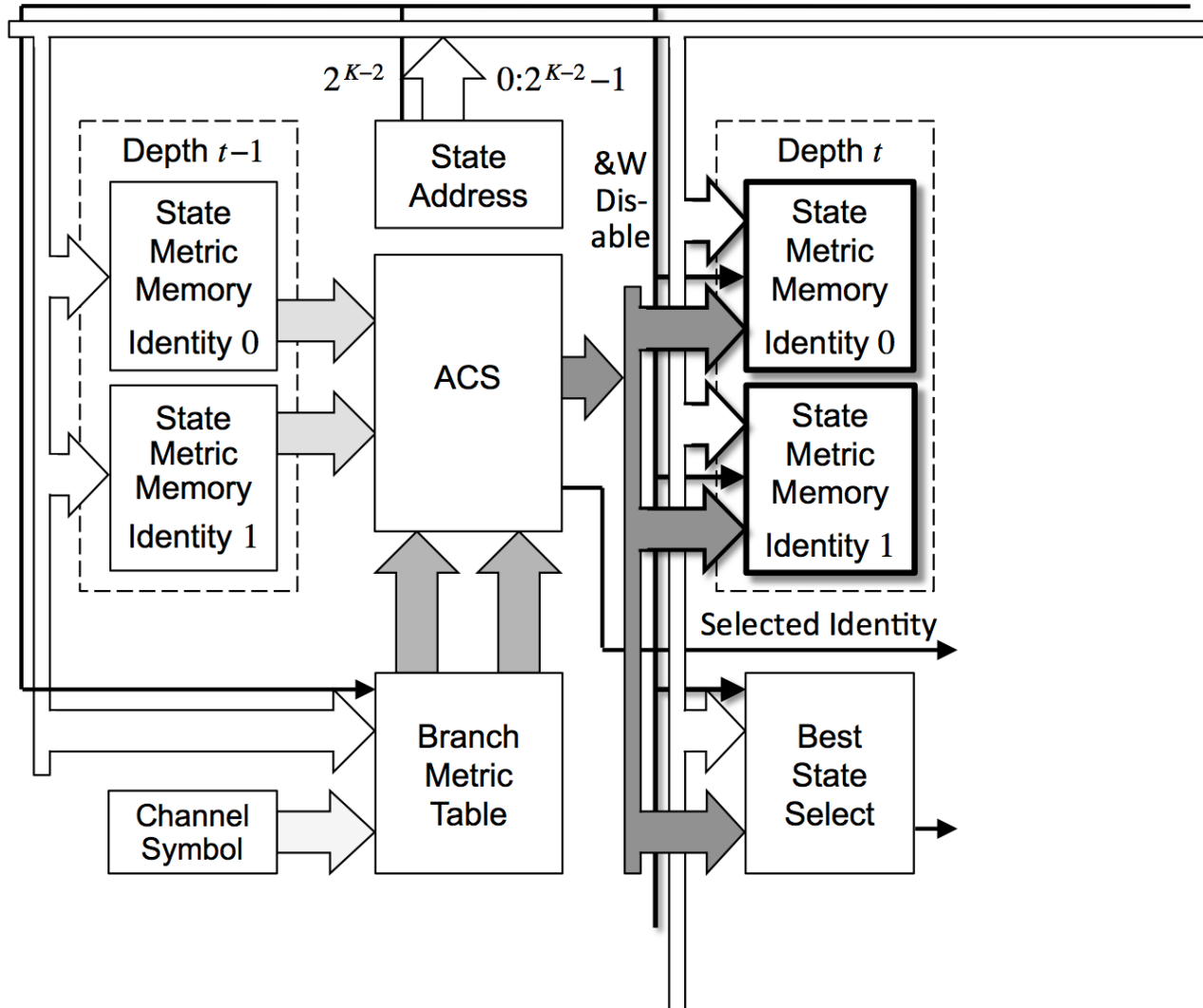
# Building Up a Viterbi Decoder, 4

State Info into ACS are Metrics from Two Halves of State Space



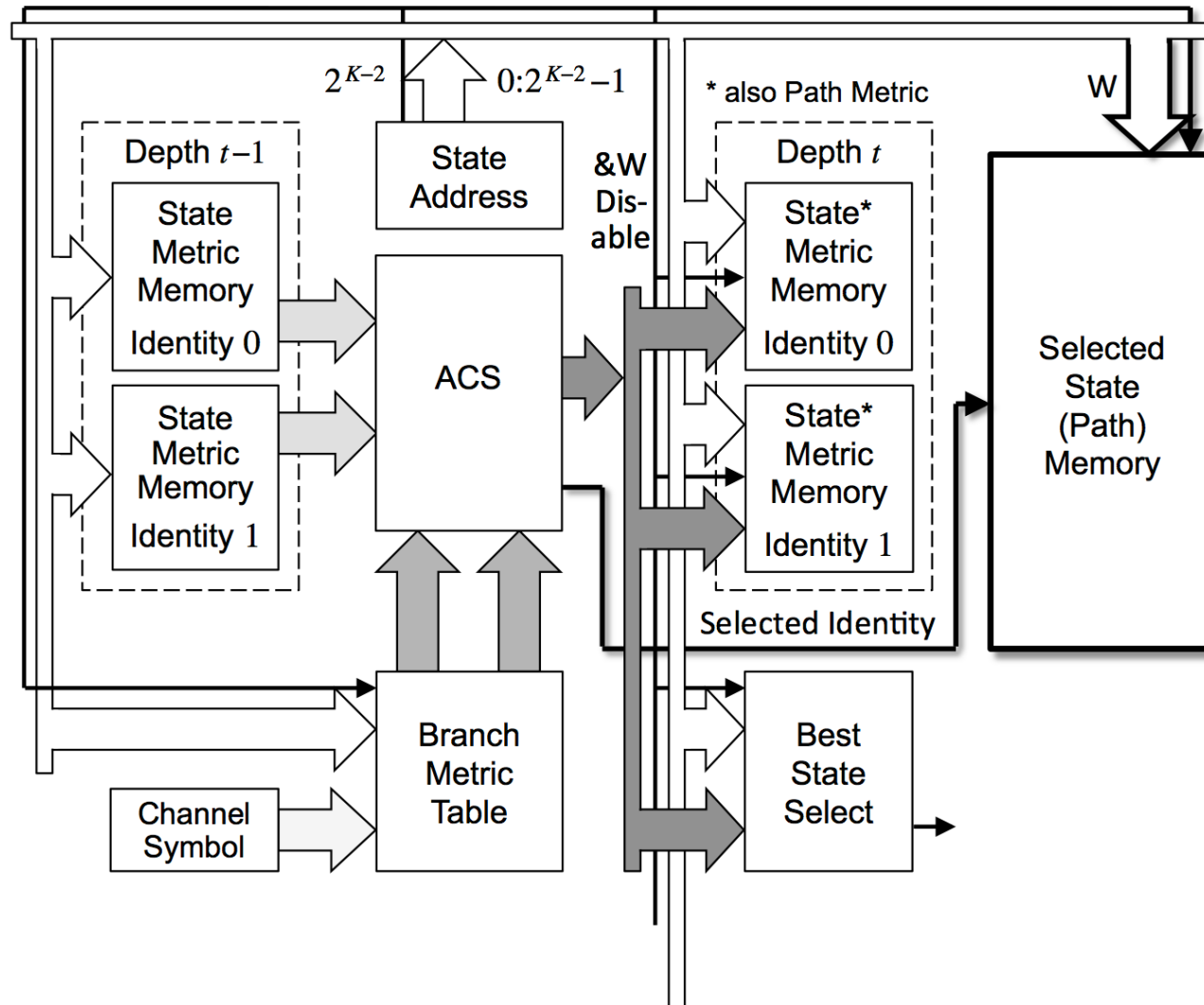
# Building Up a Viterbi Decoder, 5

State Info out of ACS = Updated Metrics of the Two Next Halves



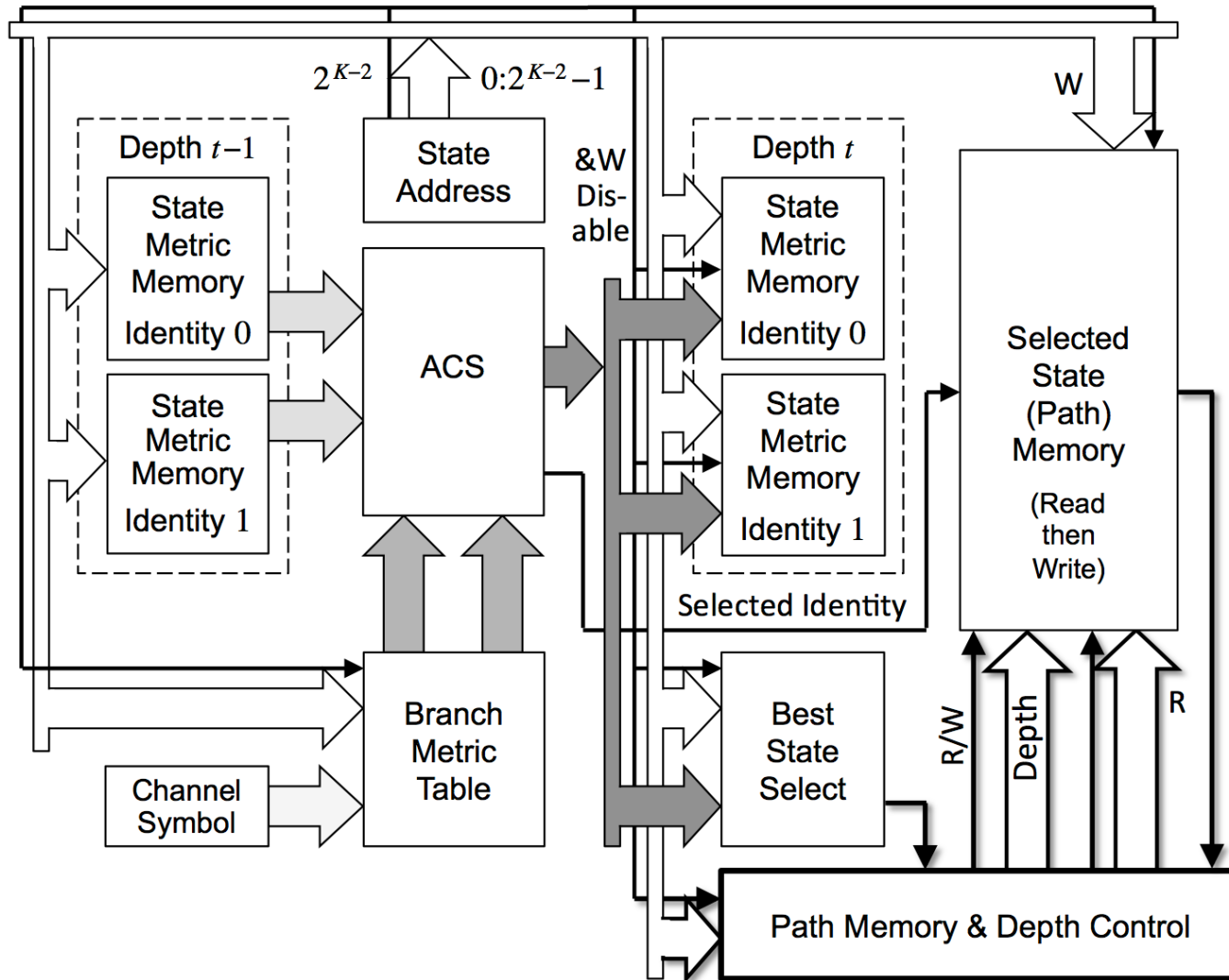
# Building Up a Viterbi Decoder, 6

Selected Identity (Bit) of Each State is Saved over Many Depths



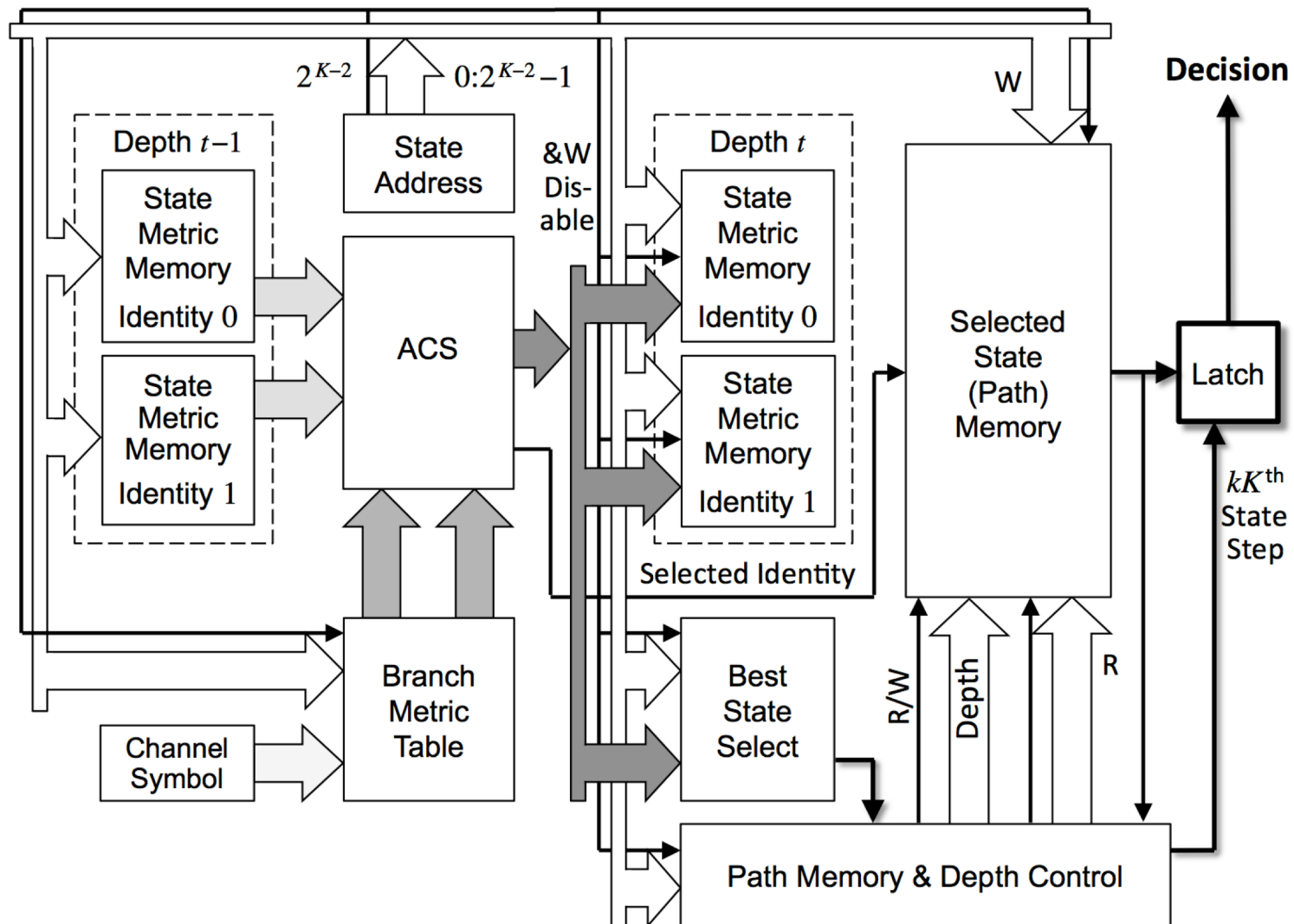
# Building Up a Viterbi Decoder, 7

Path Memory Write-once & Read-backs Requires Control



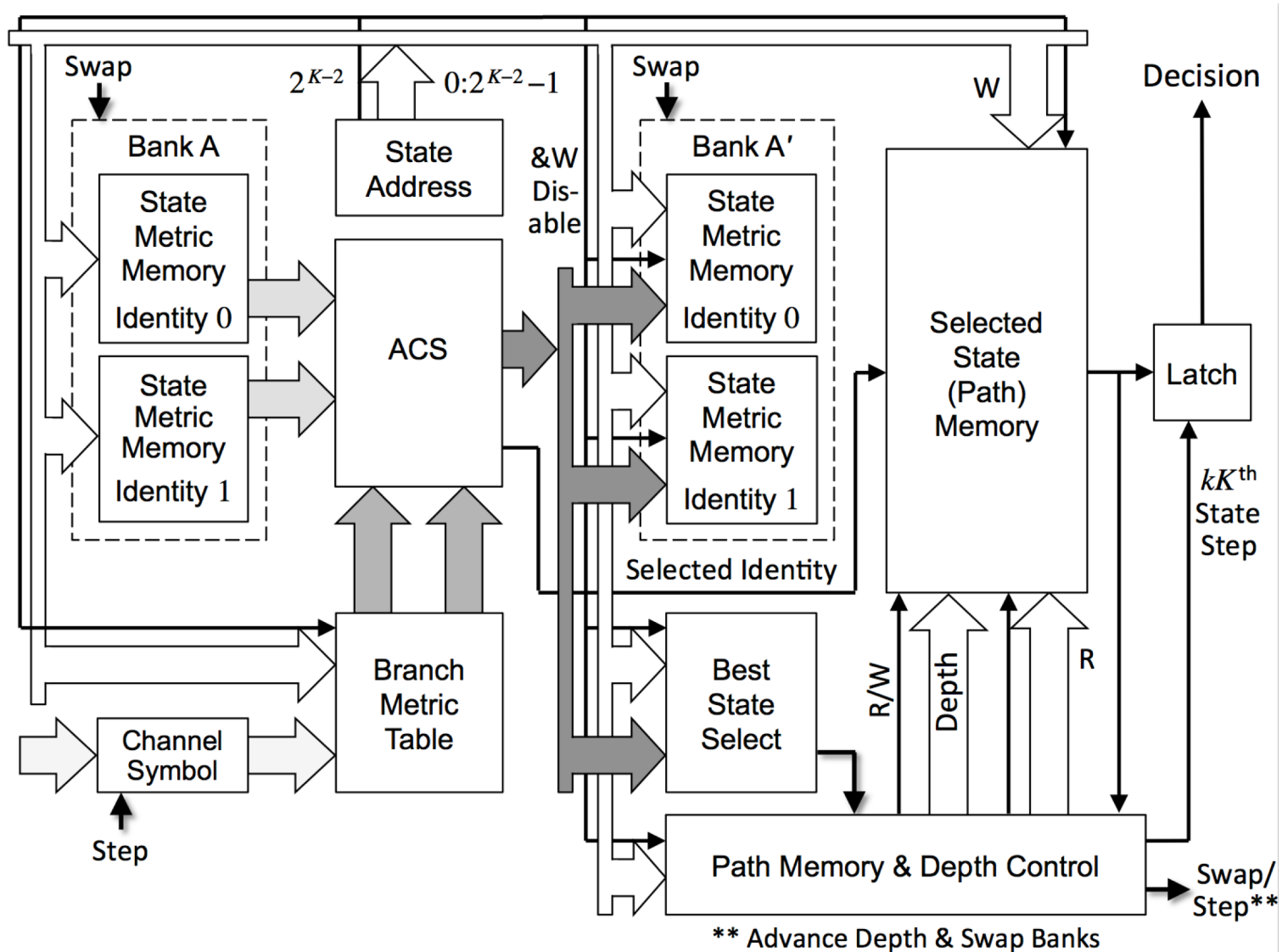
# Building Up a Viterbi Decoder, 8

Decisions are the oldest Read-backs from Path Memory



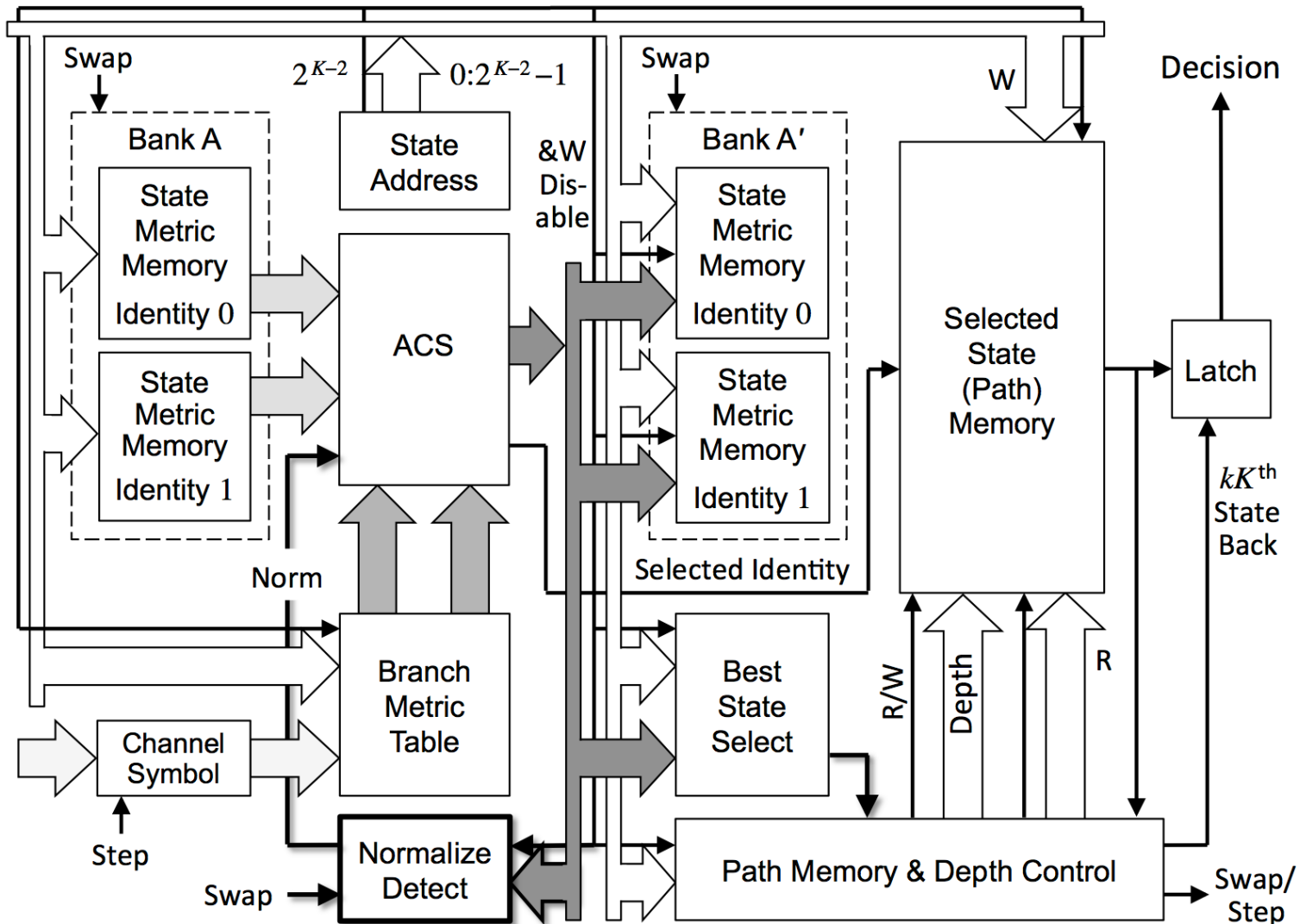
# Building Up a Viterbi Decoder, 9

At Each New Depth, State Memory Banks are Simply Swapped



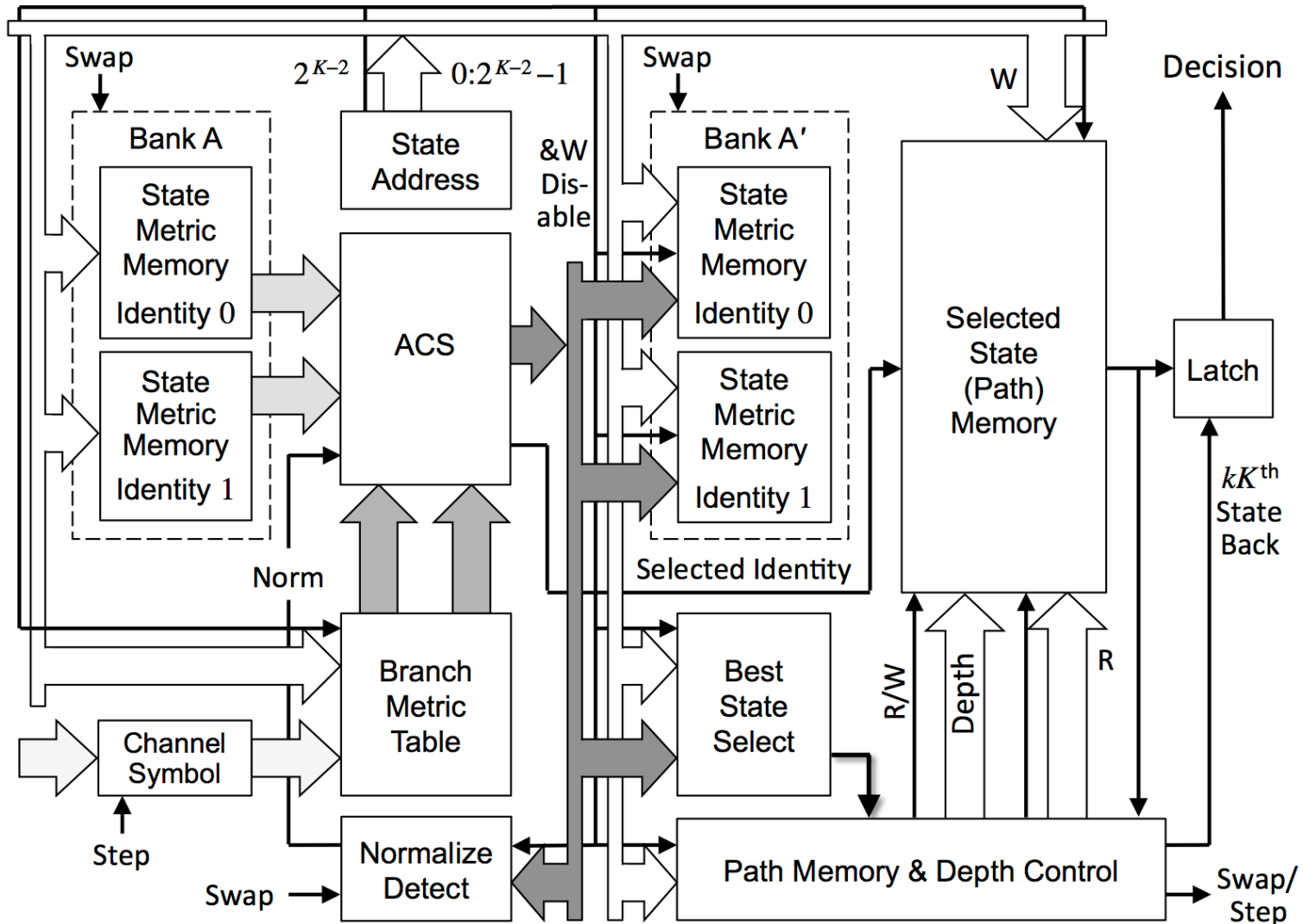
# Building Up a Viterbi Decoder, 10

To Keep Math within Limits, Normalizations Occur as Needed



# Building Up a Viterbi Decoder, Complete!

While I/O & Support is yet to be Added, Basic Ops are Complete





# Finally, Returning to Bounds

## Trace Back and Critical Runlength

- We alluded earlier to Viterbi's Fig. 5.6 in his 1979 text
- Chasing memory truncation, Viterbi sought to bound the average  $P_e$  due to a longest *unmerged path*,  $k$  beyond  $K$
- Directly involving error runlengths (error cluster lengths)
  - He needed to use a *critical length*  $\lambda_{\text{crit}}$  to maximize this bound

$$\overline{\Pi_k(j)} < (2^b - 1) e^{-N E(R, \lambda)} = (2^b - 1) 2^{-Kb(1+\lambda)E(R, \lambda)/R}$$

where  $E(R, \lambda) = E_0(\rho) - \rho E'_0(\rho)$ ,  $0 \leq \rho \leq 1$ ,  $E'_0(1) \leq \frac{\lambda R}{1 + \lambda} < C$

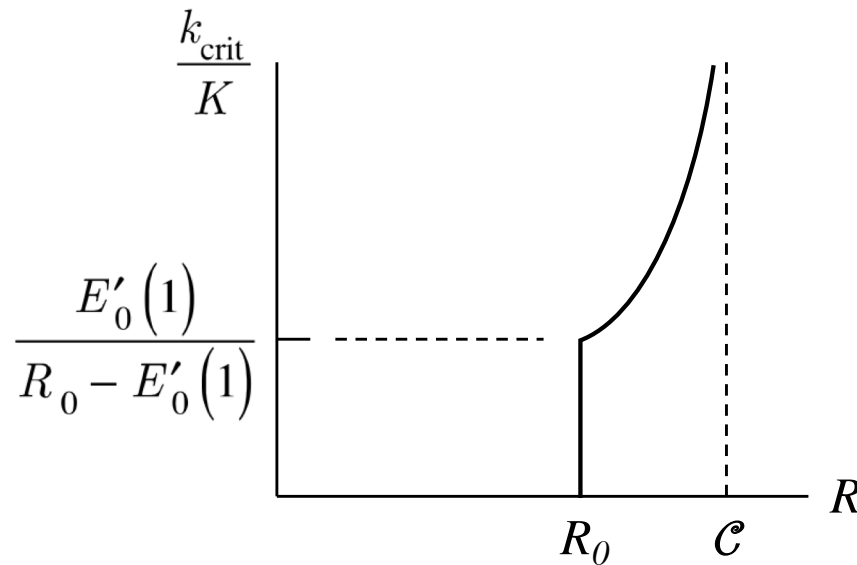
and  $E_0(\rho) = \max_{\mathbf{q}} E_0(\rho, \mathbf{q})$   $\lambda_{\text{crit}} = \frac{k_{\text{crit}}}{K} = \frac{\rho E'_0(\rho)}{E_0(\rho) - \rho E'_0(\rho)}$ ,  $0 \leq \rho \leq 1$

---

# The Importance of Fig. 5.6 (Viterbi & Omura)

## Trace Back and Critical Runlength

- So here it is, the  $k_{\text{crit}}$  - what does it mean? It means ...
- A convolutional code with ML decoding can approach capacity *iff* decoding can span an *expected longest* error cluster runlength



*beyond  
all the codes  
we knew in 1979  
(presaging turbo codes)*

- It also suggests, by its very existence, that there is such a way!

Thank You  
Design Great Stuff