False Sharing tritt auf, wenn mehrere Threads unterschiedliche Variablen verwenden, die sich auf derselben Cache Line befinden. Dadurch führt die Arbeit der Threads, obwohl sie logisch unabhängig voneinander ist, zu unnötiger Synchronisation und Invalidierung der Cache Lines, was die Leistung verringert. Dies passiert aufgrund der Cache-Kohärenz-Protokolle, die sicherstellen, dass jede Kopie einer Cache Line auf den verschiedenen Cores synchron bleibt.

**Ablauf in der Abbildung** (False sharing.pdf)**:**

1. **Speicherzugriff von Core 2 (Thread 2)**:
   * In Schritt **1** lädt **Thread 2** die Daten aus dem RAM in seinen Cache. Es wird eine ganze Cache Line geladen, die mehrere Blöcke enthält. In der Abbildung sind diese Blöcke in verschiedenen Farben dargestellt, wobei **rot** und **schwarz** markiert sind. Diese Cache Line wird jetzt von **Thread 2** im Cache von **Core 2** gehalten.
2. **Speicherzugriff von Core 1 (Thread 1)**:
   * **Thread 1** auf **Core 1** greift auf den **roten** Block zu, der sich auf derselben Cache Line befindet wie die Daten, die **Thread 2** gerade im Cache hält. Da die Daten im RAM geändert wurden, wird die Kopie im Cache von **Core 2** jetzt ungültig gemacht.
3. **Invalidierung und Neuladen der Cache Line**:
   * In Schritt **3** wird die gesamte Cache Line von **Thread 2** ungültig, und **Core 2** muss die gesamte Cache Line erneut laden, um den aktuellen Zustand der Daten zu haben. Das passiert, obwohl **Thread 1** eigentlich nur einen kleinen Teil der Cache Line verändert hat.
4. **Konsequenz: Cache-Kohärenz-Protokoll**:
   * Dieser Vorgang wird durch das **Cache-Kohärenz-Protokoll** erzwungen, das sicherstellt, dass alle Cores immer konsistente Daten haben. Da aber beide Threads unterschiedliche Daten bearbeiten wollen, die sich zufällig auf derselben Cache Line befinden, kommt es zu einem ständigen Invalidieren und Neuladen der Cache Line. Diese unnötige Synchronisation verursacht erhebliche Performance-Einbußen.

**Lösung**: Um False Sharing zu vermeiden, kann man versuchen, sicherzustellen, dass die von verschiedenen Threads verwendeten Variablen in unterschiedlichen Cache Lines liegen. Dies kann durch **Padding** (Hinzufügen von zusätzlichen Bytes) oder durch eine bewusste Speicherplatzaufteilung erreicht werden.

Die Abbildung "Performance gains after Moore’s law ends" zeigt, dass trotz der Verlangsamung von Moores Gesetz immer noch Leistungssteigerungen in der Computertechnologie möglich sind. Obwohl die exponentielle Zunahme der Transistoranzahl auf Chips aufgrund physikalischer Grenzen abnimmt (also ab einem Punkt werden die Transistoren einfach viel zu klein), gibt es andere Wege, die Leistung zu verbessern.

Durch Fortschritte in der **Architektur** können effizientere Prozessoren entwickelt werden. Zum Beispiel kann die Parallelverarbeitung durch Multicore-Prozessoren oder viele Kerne auf einem Chip gesteigert werden. Auch spezialisierte Hardwarebeschleuniger wie GPUs (Graphics Processing Units) für Grafikberechnungen oder TPUs (Tensor Processing Units) für maschinelles Lernen tragen zur Leistungssteigerung bei. Verbesserungen in der Speicherarchitektur, wie schnellere Cache-Systeme oder effizientere Datenbusse, erhöhen ebenfalls die Effizienz des Systems (dazu sind sie sehr teuer, aber abgesehen davon).

Verbesserungen bei **Algorithmen** ermöglichen es, Aufgaben mit weniger Rechenleistung zu bewältigen, was unabhängig von der Hardware zu Leistungsgewinnen führt (man braucht also mehr Programmierer, die auf Effizienz achten). Zum Beispiel können neue Sortieralgorithmen die Daten schneller ordnen oder optimierte Suchalgorithmen den Zugriff auf Informationen beschleunigen. Auch Algorithmen, die speziell für parallele Verarbeitung entwickelt wurden, können die vorhandene Hardware besser ausnutzen.

Auch optimierte **Software** und Compiler tragen dazu bei, die vorhandene Hardware besser auszunutzen. Durch fortschrittliche Compiler-Techniken können Programme effizienteren Maschinencode erzeugen, der die Hardware-Ressourcen optimal nutzt. Software-Optimierungen können Engpässe beseitigen und die Ausführungsgeschwindigkeit erhöhen. Zudem ermöglichen es virtuelle Maschinen und Laufzeitumgebungen, Programme plattformübergreifend effizienter auszuführen.

Es kann also eine enge **Systemintegration** von Hardware und Software, angepasst an spezifische Anwendungen, zusätzliche Effizienzsteigerungen bringen. Durch das Co-Design von Hardware und Software können Systeme geschaffen werden, die genau auf bestimmte Aufgaben zugeschnitten sind, wie zum Beispiel in eingebetteten Systemen oder spezialisierten Servern. Solche maßgeschneiderten Lösungen können die Leistung und Effizienz erheblich steigern, da alle Komponenten optimal zusammenarbeiten.